# A Generic Flexible and Scalable Method for Using Evolutionary Algorithms in Cluster Computing Environments

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

M.Sc. Hatem Khalloof

aus Ain Aljern

Tag der mündlichen Prüfung: 16.06.2023
Erster Gutachter: Prof. Dr. Veit Hagenmeyer
Zweiter Gutachter: T.T.-Prof. Dr. Pascal Friederich

*An investment in knowledge pays the best interest*
— Benjamin Franklin

# Abstract

Recently, the complexity of optimization problems has dramatically increased leading to non-linear, multimodal, constrained and non-convex search spaces. Generally, deterministic or conventional optimization methods cannot tackle such problems without simplifications which in turn are not trivial tasks. Among others, population-based metaheuristics – such as evolutionary algorithms – are convincing alternatives to solve complex and large-scale optimization problems. They make hardly any demands on the formulation of the optimization problem since they do not require convexity, linearity, continuity or derivability. They are able to escape from local optima and handle objective functions with noise. Additionally, population-based metaheuristics have a generic implementation which facilitates their adaptation to solve any new optimization problems. These beneficial traits do not exclude some drawbacks such as the absence of guarantee for finding the global optimum and the high computational demands for solving complex and large-scale problems in a timely manner. To overcome these limitations, it is of great importance to leverage the advantage of the inherent parallel nature of population-based metaheuristics and their ability to formulate hybrid methods along with other algorithmic approaches.

In the context of parallelizing and hybridizing population-based metaheuristics, the interest in using robust and modern software technologies for exploiting the enormous computational capabilities of modern hardware such as clusters and cloud environments has increased. But despite that, there are still some open challenges to be faced. For example, which software design can be used to facilitate the parallelization process of already developed population-based metaheuristics and increase the flexibility to hybridize them with other algorithms to solve more complex tasks in cluster computing environments.

In the present thesis, a generic, flexible and scalable method for using existing population-based metaheuristics such as evolutionary algorithms in cluster computing environments is conceptualized and developed. To ensure more efficiency in parallelizing and hybridizing evolutionary algorithms, our solution is developed in a highly modular manner in contrast to state-of-the-art monolithic complex software architectures. In this solution, we handle the problem of parallelization of evolutionary algorithms by proposing two methods. For both of them, we introduce a new perspective to realize and deploy several parallel models of evolutionary algorithms in cluster computing environments by distinguishing between functionalities related to the evolutionary algorithms and the ones related to the parallelization models. We decouple the basic building blocks of each functionality and encapsulate them in separate small and autonomous software; known as service. This enables the developers of evolutionary algorithms to easily combine the building blocks to form one of the well-known parallelization models. While the first method focuses on mapping existing evolutionary algorithms to the basic parallelization models such as the Global Model and the Coarse-Grained Model, the second one combines two or more of the basic parallelization models in a hierarchical form to enhance the parallel performance by reducing the disadvantages of each model and leveraging the advantages.

For enhancing the performance and the applicability of the evolutionary algorithms on a wide range of application scenarios, a hybridization process between existing evolutionary algorithms and other algorithms in cluster computing environments is required. To facilitate such hybridization, further development of our solution is achieved whereby two new hybridization approaches based on machine learning techniques for the partial seeding of the initial populations of evolutionary algorithms are proposed. In the present thesis, we introduce simple mechanisms to facilitate the collaboration of evolutionary algorithms with other algorithms and tools like simulators and forecasting frameworks. Our benchmark experiments on real-world optimization problems indicate that the proposed solution is promising not only in increasing the scalability and enhancing the performance of evolutionary algorithms but also in extending the applicability of them, hence opening new perspectives for applying evolutionary algorithms on a wide spectrum of real-world optimization problems.

# Zusammenfassung

In letzter Zeit hat die Komplexität von Optimierungsproblemen drastisch zugenommen, was zu nichtlinearen, multimodalen, komplex beschränkten und nicht konvexen Suchräumen führt. Im Allgemeinen können deterministische oder konventionelle Optimierungsmethoden solche Probleme nicht ohne Vereinfachungen lösen, was wiederum eine komplexe Aufgaben darstellt. Populationsbasierte Metaheuristiken - wie evolutionäre Algorithmen - sind überzeugende Alternativen zur Lösung komplexer und großskaliger Optimierungsprobleme. Sie stellen kaum Anforderungen an die Formulierung des Optimierungsproblems, da sie keine Konvexität, Linearität, Stetigkeit oder Differnzierbarkeit erfordern. Sie sind in der Lage, lokale Optima zu vermeiden und Zielfunktionen mit Rauschen zu behandeln. Darüber hinaus stehen populationsbasierte Metaheuristiken über eine generische Implementierung zur Verfügung, die ihre Anpassung zur Lösung beliebiger neuer Optimierungsprobleme erleichtert. Diese vorteilhaften Eigenschaften sind jedoch nicht ohne Nachteile, wie der Verlust der Garantie für das Auffinden des globalen Optimums und der hohe Rechenaufwand für die zeitnahe Lösung komplexer Probleme. Um diese Einschränkungen zu überwinden, ist es von großer Bedeutung, die inhärente Parallelität von populationsbasierten Metaheuristiken und ihre Fähigkeit, hybride Methoden zusammen mit anderen algorithmischen Ansätze zu formulieren, zu nutzen.

Im Zusammenhang mit der Parallelisierung und Hybridisierung von populationsbasierten Metaheuristiken hat das Interesse am Einsatz robuster und moderner Softwaretechnologien zur Nutzung der gestiegenen Rechenkapazitäten moderner Hardware wie Cluster und Cloud-Umgebungen gewachsen. Trotz dieser Entwicklungen gibt es immer noch offene Herausforderungen, wie man den Parallelisierungsprozess bereits entwickelter populationsbasierter Metaheuristiken in Cluster-Computing-Umgebungen und die Hybridisierung mit anderen Algorithmen erleichtern kann und wie man die parallele Leistung der evolutionären Algorithmen verbessern kann.

In dieser Dissertation wird ein generisches, flexibles und skalierbares Verfahren zur Nutzung bestehender populationsbasierter Metaheuristiken im Allgemeinen und evolutionärer Algorithmen im Besonderen in Cluster-Computing-Umgebungen konzipiert und entwickelt. Um mehr Effizienz bei der Parallelisierung und Hybridisierung von evolutionären Algorithmen zu gewährleisten, ist unsere Lösung im Gegensatz zu modernen monolithischen komplexen Softwarearchitekturen hochgradig modular aufgebaut. In unserer Lösung behandeln wir das Problem der Parallelisierung von evolutionären Algorithmen durch die Einführung zweier Methoden. In beiden Methoden stellen wir eine neue Perspektive für die Realisierung und den Einsatz verschiedener paralleler Modelle evolutionärer Algorithmen in Cluster-Computing-Umgebungen vor, indem wir zwischen Funktionalitäten, die mit den evolutionären Algorithmen zusammenhängen, und solchen, die mit den Parallelisierungsmodellen zusammenhängen, unterscheiden. Wir entkoppeln die grundlegenden Bausteine jeder Funktionalität und kapseln sie in separater und autonomer Software, die als Service bezeichnet wird. Dies ermöglicht es den Entwicklern von evolutionären Algorithmen, die Bausteine einfach zu kombinieren, um eine der grundlegenden Parallelisierungsmodelle umzusetzen. Während sich die erste Methode darauf fokussiert, bestehende evolutionäre Algorithmen auf eines der grundlegenden Parallelisierungsmodelle wie das Global Modell und das Coarse-Grained Modell abzubilden, fokussiert sich die zweite Methode darauf, zwei oder mehr der grundlegenden Parallelisierungsmodelle in einer hierarchischen Form zu kombinieren, um die parallele Leistung des EA zu verbessern, indem die Nachteile jedes Modells reduziert und die Vorteile genutzt werden.

Um die Leistung und die Anwendbarkeit der evolutionären Algorithmen in einem breiten Spektrum von Anwendungsszenarien zu verbessern, ist ein Hybridisierungsprozess zwischen bestehenden evolutionären Algorithmen und anderen Algorithmen in Cluster-Computing-Umgebungen erforderlich. Um eine solche Hybridisierung zu ermöglichen, wird unsere Lösung weiterentwickelt, wobei zwei neue Hybridisierungsansätze für das partielle Seeding der initialen Population von evolutionären Algorithmen durch den Einsatz von maschinellen Lernen untersucht werden. In dieser Arbeit stellen wir Mechanismen vor, die die Zusammenarbeit von evolutionären Algorithmen mit anderen Algorithmen und Werkzeugen wie Simulatoren und Vorhersage-Frameworks erleichtern. Unsere Benchmark-Experimente mit realen Optimierungsproblemen zeigen, dass die vorgeschlagene Methode nicht nur die Skalierbarkeit und die Leistung von evolutionären Algorithmen erhöht, sondern auch deren Anwendbarkeit, indem sie neue Perspektiven für die Anwendung evolutionärer Algorithmen auf ein breites Spektrum von realen Optimierungsproblemen eröffnet.

# Acknowledgement

The research presented in this dissertation was conducted at the Institute for Automation and Applied Informatics (IAI) of the Karlsruhe Institute of Technology. Attaining the present dissertation has been a remarkable journey and I would like to express my utmost gratitude to all those who have supported me along this journey.

First and foremost, I would like to express my genuine heartfelt appreciation to my advisor, Prof. Dr. Veit Hagenmeyer for granting me the opportunity to pursue my Ph.D. research under his supervision in a highly competitive working environment at IAI. Very special thanks goes to the co-referee Prof. Dr. Pascal Friederich, for accepting the role of reviewing my thesis.

I would like to express my appreciation and gratitude to Dr. Clemens Düpmeier for his inspiring ideas, continuous support and constructive advice from the beginning of my research journey. His guidance has been invaluable in shaping my work. I would also like to express my thanks to Dr. Wilfried Jakob for his unwavering support, guidance and collaborative efforts throughout the research period. Working closely with them has allowed me to enhance my understanding in the field of evolutionary algorithms, parallel computing and software engineering.

I want to express my appreciation to all the students that I had the pleasure of working with (in order of appearance): Phil Ostheimer, Mohammad Mohammad, Seifeddine Mazigh, Sergen Ciftci, Saina Khalili and Ragheed Khalouf. Mentoring you has been the most rewarding aspect of my role at IAI.

I also wish to thank my longtime friend (from bachelor to Ph.D.) and best office mate, Dr. Shadi Shahoud. This success would not have been achieved without your support, your great mind, and our sensitive and fruitful discussions throughout 17 years. Thank you for patiently answering all my questions and providing valuable feedback. Thank you for reviewing this dissertation. Your expertise and willingness to assist have been instrumental in the development of this research. I would also like to express my gratitude to my colleagues at IAI for the wonderful five years we've shared, particularly Dominique Sauer, Eric Braun, Richard Jumar, Rafael Poppenborg, Christian Schmitt, Claudia Greceanu, Dr. Thorsten Schlachter, Christina Griess, Jianlei Liu, Richard Lutz, Malte Chlosta, Jannik Sidler and Dr. Kevin Förderer.

I would like to convey special thanks to my parents, brothers and sister. The supreme gratitude is to my wise mother Najeha Khalloof who is the reason for everything I have become. You have always been by my side, endlessly supporting and caring for me throughout my whole life. I am eternally grateful.

The ultimate thanks goes to my father Elias Khalloof who was always my safety foundation, and who has always worked very hard to be a successful man. Rest in peace legend.

The last paragraph is dedicated to my wife, Zeina Btrous, who has provided me with power, as well as moral and emotional support in every aspect of my life. Thank you for standing by me during times of stress and difficulties and for your patience in the last year. Each day, you bring happiness into my life. I am grateful for your presence and for the blossoming of my soul since you entered my life. Thank you, sweetheart. Thank you, my small angel Kayla, for bearing your busy Dad during your first year. Your laughs erase tiredness. You are both my ultimate inspiration.

Karlsruhe, Juni 2023 Hatem Khalloof

# Contents

# List of Figures

# List of Tables

# 1. Introduction

## 1.1. Motivation

Nowadays, optimization problems are a part of many real-world applications such as logistics, production, power grids, finance and nature sciences, e.g., [22, 43, 59]. Most of these problems are of complex nature due to, e.g., non-linearities, multi-objectives, discontinuities or restrictions. Furthermore, the majority of them have a large-scale search space since they extensively arise from different modern industrial applications which contain thousands of decision variables and multiple constraints [3]. For solving optimization problems, there are two main methods [3], namely exact and heuristic methods as shown in Figure 1.1. Exact methods are often not applicable to find the optimal solution for complex and large-scale optimization problems in a timely manner. They are computationally expensive since the execution time exponentially increases according to the problem size [146]. Using specific heuristic methods tends to be faster than conducting the exact methods but they are not considered generic techniques. In other words, their designs are not general enough to be adapted with minimal effort for solving other similar optimization problems [143]. Unlike exact and simple heuristic methods, metaheuristics are more generic optimization techniques that offer good solution quality while maintaining a reasonable runtime. Metaheuristics combine different search concepts for exploring and exploiting the search space to find the near-optimal solutions or even the optimal ones efficiently [25, 67].

Among metaheuristics, Evolutionary Algorithms (EAs) are becoming increasingly vital optimization techniques for solving complex optimization problems in many fields, e.g., industry, science, healthcare and sports among others [17, 111, 150, 46]. This popularity arises due to their design flexibility and ability to solve a wide range of optimization problems in a relatively short time, finding good local optima or even global ones [142]. EAs are population-based random search algorithms that mimic the mechanisms of biological evolution by generating a population representing a set of possible solutions. Each individual (solution) undergoes some evolutionary procedures. The main steps of EAs can be summarized as follows. Initially, an initial population is generated. Then, the fitness of each individual is calculated. Afterwards, the principle of natural selection, namely 'survival of the fittest' is applied in order to select the parents of the next generation based on their fitness values. After that, the genetic operators, namely crossover and mutation are applied to the selected parents to generate offspring, on which the fitness evaluation is again applied. This procedure iteratively continues until the fitness evaluation reaches a predefined value or another termination criterion is met. Despite the clear advantages of applying EAs in a wide range of application areas, EAs suffer from scalability issues and

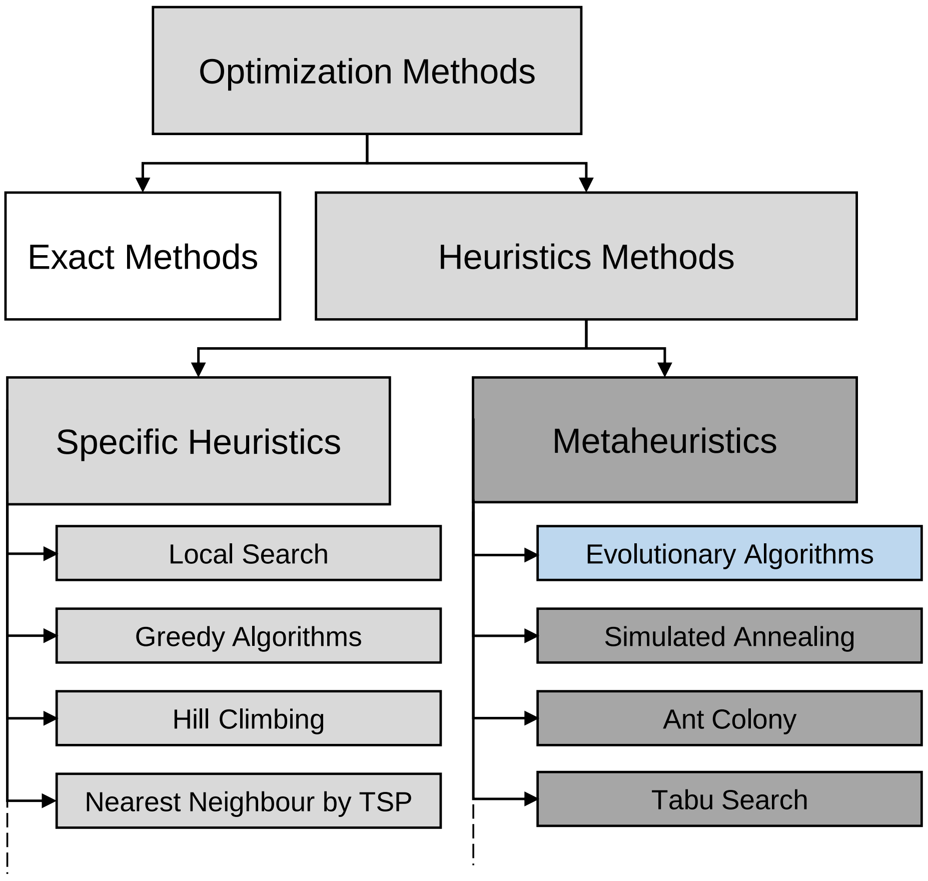


**Figure 1.1.:** Classification of optimization methods adapted from [174, 71]

admittedly need a lot of computational power to find an adequate solution for a complex and large-scale optimization problem in a timely manner. This is mainly due to the fact that the individuals are evolved in an iterative process which requires a high number of function evaluations. According to the application domains, these evaluations can reduce the performance of the EAs to inappropriate levels, especially when complex calculations are required such as simulations. A further common drawback of EAs is the premature convergence, in which EAs cannot generate any more solutions with better quality than the actual ones [58]. This occurs since the genetic operators lose the capability to enhance the population by generating offspring better than the parent. To overcome these drawbacks and increase the usage of EAs, two approaches are proposed, namely the parallelization approach and the hybridization approach as shown in Figure 1.2.

Parallelization of EAs has attracted the attention of researchers as a solution for treating the challenge of scalability and preventing premature convergence. Cantú-Paz [35] proposed and investigated three basic parallel models, namely the Global Model (Master-Worker Model), the Fine-Grained Model (Grid Model) and the Coarse-Grained Model (Island Model) featuring the inherent parallelism of EAs as shown in Figure 1.2. The Global Model distributes the evaluation effort of a population over several workers, e.g., CPUs to reduce the amount of execution time. The master holds the frequently panmictic population and performs the genetic operators on it. In the Coarse-Grained Model, the whole population is

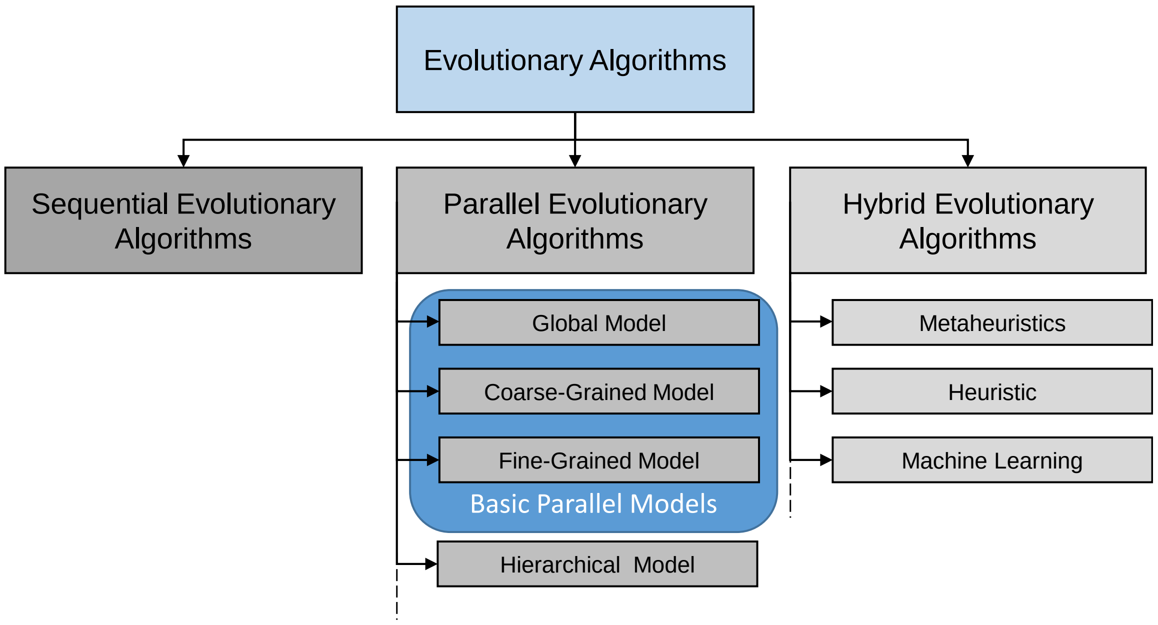


**Figure 1.2.:** Evolutionary algorithms and their implementations

divided into multiple subpopulations (so-called demes) where each deme forms an island. A sequential EA is locally applied within an island to evolve the deme. Periodically, the islands exchange some promising individuals which are known as migrants according to a predefined migration policy that defines, e.g., which migrants are selected and when migrants are exchanged among the islands. A communication topology defines the neighbors of each island to determine among which islands the migration occurs. In the Fine-Grained Model, the population is spatially structured in, e.g., a grid or a ring structure restricting the spread of genetic information to the neighbors. Due to the overlapping between the neighbors, genetic information can slowly spread over the population. Each model emerging from these three basic models increases the performance of EAs to a reasonable level and reduces the risk of premature convergence considerably. However, each one has its own drawbacks which limit its scalability and applicability for solving complex and large-scale optimization problems. This leads us to add a further parallelization level to one model by combining it with other models to form the Hierarchical Model [35]. Generally, the Hierarchical Model hybridizes the Coarse-Grained Model on a high level with some characteristics of other parallelization models on low levels.

Over the years, several software architectures and frameworks have been introduced, e.g., [14, 62, 145, 52, 170, 73, 8, 91, 45, 33, 130, 183, 2, 161, 165, 136, 65, 66, 131, 163, 164] to accelerate the execution of EAs for solving complex and large-scale optimization problems. These frameworks mainly use three parallel computing models and technologies, namely conventional parallel computing model, e.g., RPC (Remote Procedure Call) [8, 33, 145, 62], Big Data technologies such as Hadoop and Spark [91, 183, 165, 45, 52, 170, 136] and modern software technologies, namely microservices and cloud technologies [131, 161, 130, 163, 164, 73, 66]. The majority of them, except the following works [163, 164, 73, 66] have a classical monolithic software architecture which decreases their modularity, usability and maintainability leading to a complex and tightly coupled system by adding new functionalities. In other words, they do not provide a generic and modular software architecture to realize and deploy any parallel model of EAs from the three basic ones. Furthermore, enhancing the performance of the parallel EAs by, e.g., combining two or more basic parallel models in a hierarchical form is also limited. Obviously, they do not support the integration of existing EAs or the cooperation with other tools such as forecasting frameworks and simulators. Hence, designing and developing a highly modular, generic, flexible and scalable software architecture for limiting the aforementioned drawbacks by facilitating the usage of EAs in a high-performance computing environment – like a cluster – still represents a challenge.

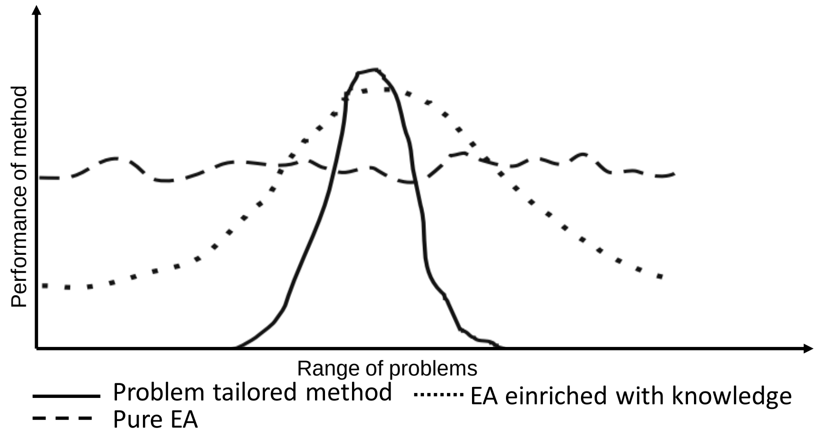


**Figure 1.3.:** Effect of enriching evolutionary algorithms with knowledge on the performance taken from [132]

Another approach to increase the performance of EAs for finding an adequate solution in a reasonable time is hybridization as shown in Figure 1.2. As with any other optimization algorithm, EAs must adhere to the ‘No Free Lunch Theorem’ [188] which states that a given optimization algorithm cannot be applied to all optimization problems in an equal manner, e.g., without exhibiting some sort of performance problem or having problems in finding an appropriate solution. Hybridizing EAs with other algorithmic approaches, e.g., metaheuristics, heuristic, Machine Learning (ML) techniques or deterministic algorithms forming Hybrid Evolutionary Algorithms (HEAs) represents a promising solution for enhancing the performance of EAs. Indeed, the hybridization of EAs not only improves the solution quality but also accelerates the search process, opening new perspectives to apply EAs on a wide spectrum of real-world optimization problems. The hybrid approach of EAs enables their use in fields where pure EAs, exact methods and heuristics provide inadequate solutions or need too much computational effort to reach an acceptable solution quality, e.g., [16, 97, 112]. There are many ways to hybridize EAs such as incorporating problem-specific knowledge to generate the initial population, e.g., [177, 180, 1, 32, 97, 110, 175, 120], approximating the fitness values of individuals, e.g., [124, 186, 155, 192, 84] or exploiting problem-specific knowledge to apply the generic operators, e.g., [76, 75, 16, 173]. For example, combining EAs with problem-specific heuristics can directly enhance the global performance of pure EAs for the given type of problem [132] as shown in Figure 1.3. For promoting such performance enhancement, a flexible and generic software solution to facilitate the hybridization of existing EAs at any step with other algorithms is required. Combining the beneficial traits of the parallelization and hybridization of EAs in a cluster computing environment without the need to rewrite or redesign the considered EA is still an open issue needs to be solved. In other words, the existing software approaches do not introduce a simple mechanism to seamlessly achieve such a combination.

## 1.2. Research Questions

With emerging modern hardware such as clusters and cloud environments, the interest in exploiting their enormous computational capabilities to speed up EAs has grown dramatically. To this end, many methods and software solutions to increase the usage of EAs in such computing environments have been proposed. Despite the extensive developments, some open questions concerning the design of a software that allows the parallelizing and/or hybridizing of EAs in a cluster environment still need to be addressed.

**Research Question 1 [RQ1]: Which software design is adequate for parallelizing EAs according to the basic parallelization models in a cluster computing environment?**

Each parallelization model of the three basic models has its advantages and disadvantages which introduce it as a highly fitted model for a specific class of optimization problems and a less suitable one for the others. For example, the main advantage of the Global Model is the simplicity enabling cooperation with single simulators or combinations of simulators for multi-domain simulations. However, its disadvantage is that the master rapidly becomes a bottleneck. Hence, a proper model when the calculation of the fitness function evaluations is computationally expensive and the communication between the workers and master does not occur with high frequency. In the Coarse-Grained Model, the main advantages are the gained speedup and the ability to improve the global search capabilities, hence avoiding the problem of premature convergence. Its major disadvantage is that many parameters have to be set and tuned for an optimal performance. This introduces it as an adequate model when the search space is complex and has many local optima. Driven by that, It is highly required to design a scalable, modular and generic software solution that enables the realization of any model from the three basic models with little adaptation effort. Moreover, the proposed software solution has to guarantee a seamless deployment of such models in a scalable runtime environment. This provides high flexibility to choose a suitable parallelization model according to the optimization problem and to the availability of computational power, e.g., cluster resources.

**Research Question 2 [RQ2]: Which benefits can be obtained from combining two or more parallel models of EAs into one parallel solution in a cluster computing environment?**

Driven by the first question, given the advantages and disadvantages of each parallelization model from the three basic ones, why not to combine two parallelization models or more in order to eliminate the drawbacks of each one and increase the benefits? This combination enhances the parallel performance of parallel EAs and allows the application of them to complex and large-scale optimization problems. To answer this question, a scalable, flexible and modular software architecture that introduces a simple and efficient mechanism to combine two or more basic models in one hierarchical model is needed. For example, by combining the Fine-Grained Model with the Global Model, we aim at achieving a good level of parallelization with the Global Model and at the same time reduce the risk of premature convergence by using the Fine-Grained Model which maintains the genetic diversity for a long time, hence finding a good solution at a reasonable time.

**Research Question 3 [RQ3]: What are the adequate methods to facilitate the usage of existing EAs in a cluster computing environment?**

This question is raised since EA developers tend to use their stable existing EAs which have been already developed and tested over many years. However, most of these EAs are mainly implemented as monolithic and non-distributed algorithms for desktop-based applications, i.e., they are not directly integrable in a cluster or a cloud environment. The complexity of using such computing infrastructure represents a big challenge, especially for EAs developers who are coming from different scientific disciplines, e.g., mathematics, physics or others with minimal knowledge in distributed systems. To exploit the potential of modern distributed environments, existing EAs need to be redesigned and rewritten using modern parallel programming paradigms, e.g., MapReduce programming model. Hence, the main challenge lies in how to design a software solution that allows the parallelization of any existing EAs without the need to consider the issues related to the underlying computing infrastructure and to completely redesign or rewrite the EAs.

**Research Question 4 [RQ4]: What are the adequate methods to facilitate the interaction between EAs and external tools and applications (e.g., simulators)?**

The fourth question addresses the challenge of integrating EAs in a complex and collaborative system, in which they interact with external tools, components like forecasting platforms and problem-specific functionalities (e.g., simulators) for performing successful optimization tasks. The contribution of external tools is required to support EAs, e.g., by defining the implicit and explicit restrictions and other tasks. These tools are usually independently developed using several technology stacks and programming languages hindering easy cooperation with EAs and other software components. Indeed, it is of importance to design a software solution that allows high-level interactions between EAs and other tools and services enabling EAs to be a part of complex industrial applications.

**Research Question 5 [RQ5]: ]: Which software design is adequate for hybridizing existing EAs with other algorithms in a cluster computing environment?**

Combining EAs with other algorithms and methods from other research fields is essential to increase the performance of EAs and open new perspectives to use EAs in several domains. For example, EAs use an initial population of potential solutions as starting points in the solution space. This initial population is generally created at random which can lead to generate inappropriate individuals that prematurely converge to a suboptimal solution. Taking the advantage of problem-specific knowledge and ML techniques, namely supervised and unsupervised learning to assist EAs by generating the initial population increases the performance and the applicability of EAs to solve complex optimization problems.

The hybridization of existing EAs with other algorithms without the need to redesign them is not a trivial task. To address this challenge, it is noteworthy to design an extensible, modular and generic software solution that facilitates the hybridization of EAs at any step of them with other algorithms and techniques in a cluster computing environment.

## 1.3. Research Goals and Contributions

The present thesis addresses the aforementioned open questions identified within the analysis of state-of-the-art software architectures and approaches for parallelizing and hybridizing EAs. The main goal of this thesis is to enhance the performance of EAs for solving real-world optimization problems by facilitating the parallelizing and/or hybridizing of EAs with other algorithms in cluster computing environments. The contributions of this work are summarized as follows.

- Developing BeeNestOpt.IAI[1], a flexible, scalable, extendible, modular and generic software solution for enhancing and facilitating the usage of EAs in cluster and cloud environments using sophisticated, stable and reliable software technologies such as microservices, container technologies and the publish/subscribe messaging paradigm.
  - BeeNestOpt.IAI provides a highly parallel and scalable environment to parallelize EAs according to any parallel model from the basic ones with full runtime automation on a computing cluster.
  - It provides a simple and efficient mechanism to hybridize EAs with other algorithmic approaches.
  - It hides the technical aspects of the underlying computing infrastructure by realizing a web-based management and execution environment.
  - It allows EAs to be a component of a complex system, in which different tools and simulators cooperate together to solve the optimization task.
- Mapping the Global Model to BeeNestOpt.IAI. The performance test reports a low and only slight increase of the communication overhead with growing population sizes by varying the number of workers. Moreover, the execution time of EAs decreases by deploying more workers.
- Extending BeeNestOpt.IAI architecture to support the Coarse-Grained Model. The parallel performance test of the proposed software architecture with the Coarse-Grained Model shows a constant migration overhead by increasing the migration rates and a superlinear/linear speedup for complex problems fitness evaluations.

[1] The name ‘BeeNestOpt.IAI’ combines three ideas: ‘Bee’ is an individual of the population – representing a solution for an optimization problem – who works to discover the fields – search space –, the ‘Nest’ is the beenest that represents the microservices as a nest of hexagons, ‘Opt’ is the abbreviation of optimization. ‘IAI’ is the abbreviation of the name of the Institute for Automation and Applied Informatics where the proposed software solution is developed.

- Enhancing the performance of the basic parallel models by supporting the hierarchical combinations between them. We combine the Coarse-Grained Model with the Global Model to form a Hierarchical Model aiming at limiting the disadvantages and promoting the advantages of both models. This combination achieves a significant speedup, maintaining a low communication and migration overhead.
- Hybridizing EAs with machine learning algorithms to alter and/or replace some parts of an EA. Two machine learning approaches are proposed to assist EA by generating the initial population. This is achieved by seeding the initial population with either already evaluated individuals or new ones generated based on the gained knowledge from the prior evaluated individuals and the related domain-based knowledge. Both hybridization approaches have positive impacts on the final solutions quality and convergence speed.
- Performing a practical evaluation based on a real-world optimization problem, in which the applicability and the performance of the proposed software architecture are studied. The problem of scheduling Distributed Energy Resources (DERs) is considered to evaluate the scalability of the software design. The results show that the proposed software solution for parallelizing and hybridizing EAs reduces the total execution time to a convincing level while still offering a good quality solution.

## 1.4. Structure of the Thesis

The rest of the present thesis is structured as follows. Chapter 2 describes the technological as well as the background knowledge used in this work. Chapter 3 discusses research projects related to the contributions of this work. This includes existing parallelization frameworks and state-of-the-art implementations in cluster and cloud environments. Then, an extensive review of different hybridization approaches and their respective realization is introduced.

In Chapter 4, the new software architecture to parallelize EAs in cluster computing environments based on container and microservice technologies is introduced addressing the research questions [RQ1, RQ3 and RQ4]. This chapter presents the concept, the architecture of the proposed software solution and the realization of the Global Model and the Coarse-Grained Model. Then, the evaluation results regarding the parallel performance for both models are introduced.

Chapter 5 addresses the research question [RQ2, RQ3 and RQ4] for combining two parallelization modes in a hierarchical form to enhance the parallel performance of the basic parallel models. Afterward, the implementation and the evaluation results obtained by deploying the Hierarchical Model on a cluster for solving a non-convex mixed-integer multi-objective optimization problem, namely the scheduling of DERs are introduced.

Chapter 6 introduces a software solution to hybridize any part of EAs with other algorithms and techniques in a cluster computing environment answering the research question [RQ5]. Moreover, it proposes new ML hybridization approaches for assisting EAs by generating the initial population exploiting domain-based knowledge. The evaluation of such hybridization is conducted by solving the problem of scheduling distributed energy resources. Chapter 7 concludes this work and presents an overview of potential future works.

# 2. Background

In this chapter, the necessary theoretical background to understand the main contributions of this thesis is introduced. Section 2.1 gives a brief general definition of optimization problems and their classifications. Section 2.2 explains in detail EAs, their parallelization models and hybridization concepts. Section 2.3 explains some fundamentals related to the parallel computing. The definition of machine learning and its two main learning strategies, namely the supervised and unsupervised ones which are used to hybridize EAs are introduced in Section 2.4. Section 2.5 concludes this chapter by describing the technologies stack used to develop the contributions of this thesis.

## 2.1. Optimization Problems

An optimization problem can be formulated as a search problem for finding the near-optimal or the optimal solution among all possible ones to minimize (or maximize) a function called the objective function subject to some constraints. Generally, any maximization problem can be converted into a minimization problem without loss of generality. Given the objective function $f$, one can always convert it to a minimization problem, since: $max\{f(x)\} = -min\{-f(x)\}$. According to [18, 81] we can define a parameter optimization problem as follows: Given the objective function $f$

$$f : M \subseteq M_1 \times ... \times M_n \rightarrow \mathbb{R}, M \neq \emptyset \tag{2.1}$$

where $\mathbb{R}$ is the set of the real numbers, the overall goal is to find $x_{opt} \in M$ such that

$$\forall x \in M : (f(x) \leq f(x_{opt})) = f_{opt} \tag{2.2}$$

where $f_{opt}$ is called a global maximum and $x_{opt}$ is the maximum location. In contrast to a global maximum, a local maximum $\hat{f} = f(\hat{x})$ is defined by

$$\exists \epsilon > 0 \ \forall x \in M : ||x - x_{lopt}|| < \epsilon \Rightarrow F_{lopt} \geq F(x) \tag{2.3}$$

$M$ is called the solution space – or search space – of the problem. If $M$ is equal to $\mathbb{R}$, an unconstrained optimization problem is defined. That a possible solution of the optimization problem has to be an element of $M$ can be seen as a very generic form of constraint condition which reduces the overall solution domain of a function to a subset of feasible values satisfying all constraints.

$$f : M_1 \times ... \times M_n \rightarrow \mathbb{R} \tag{2.4}$$

| **Property** | **Type of optimization problem** |
|---|---|
| Objective function, constraints or search space | Linear or non-linear, uni or multimodal, continuous or discontinuous, convex or non-convex |
| Kind of decision variables | Discrete, continuous or mixed-integer (i.e., continuous and discrete) |
| Search space | Constrained or unconstrained |
| Number of objectives | Single- or multi-objective optimization |

**Table 2.1.:** Important properties of optimization problems

$M$ is therefore often also called the set of feasible values of the optimization problem. Typically, $M$ is constrained by inequality and equality constraints:

$$G_i(x) \leq 0 \ and \ H_i(x) = 0 \tag{2.5}$$

where $i, j \leq n$. The nature of the objective function $f$ and constraints $G$ and $H$ refers to the degree of linearity or non-linearity of the functions themselves. If the objective function and constraints are linear functions, the problem is called a linear optimization problem which is formed, e.g., as follows.

$$f(x,y) = ax + by, x \geq 0 \ and \ y \geq 0 \tag{2.6}$$

subject to:

$$G_i(x,y) = cx + dy \leq e \ and \ H_i(x,y) = lx + my = 0 \tag{2.7}$$

Similarly, if they are non-linear, e.g., – quadratic (e.g., $f = x^2$) or non-convex (e.g., $f = x^4 + x^3 + x^2 + 9x + 1$) – the optimization problem is classified as a non-linear optimization problem. If the set $M$ has only a discrete number of elements, the problem is called a discrete optimization problem. Such optimization problem is in principle decidable by evaluating all possible solutions but typically in such problems $M$ is very large and therefore it is not feasible to solve it by brute-force evaluation because of performance constraints. It can also be shown that some discrete optimization problems are NP-hard, e.g., integer linear programming, the traveling salesman problem, the knapsack problem or finding the max cut in a graph. If the set $M$ is uncountably infinite (e.g., a subset of a cross-product of real numbers), the optimization problem is called continuous optimization. Furthermore, if all $M_i$ consist only of integral values, we call the optimization problem pure integer programming problem. If some of $M_i$ are real numbers, a mixed integer programming problem is formulated. We can distinguish between two types of optimization problems based on the number of optima. Optimization problems that have only one optimum are called unimodal. Otherwise, they are called multimodal, i.e., if they have more than one optimum. The number of objective functions also defines the nature of the optimization problem. Optimization problems with one objective are referred to single-objective optimization problems. In contrast to that, a multi-objective problem has several objective functions that are simultaneously optimized. The above-mentioned properties of optimization problems are summarized in Table 2.1.

Surprisingly, optimization problems with small size are solvable using exact optimization methods such as linear programming, sequential quadratic programming and generalized gradient methods [158]. They become easier if they are linear, convex and continuous. However, most real-world optimization problems are large-scale non-linear, non-convex, multimodal, mixed-integer and multi-objective which makes optimization more complex. In general, such optimization problems are NP-hard and therefore exact optimization methods are not applicable, since they need exponential effort to solve them [146]. Typically, heuristics with their special class, namely metaheuristics such as Evolutionary Algorithms (EAs), simulated annealing, tabu search, ant colony optimization, particle swarm optimization and others can find adequate solutions with reasonable quality in a timely manner for such complex optimization problems. Since metaheuristics are defined as general and abstract methods, they can be applied to different optimization problems with few adaptations. EAs represent among others the most popular method to solve complex and large-scale optimization problems, e.g., [26, 133, 79].

## 2.2. Evolutionary Algorithms (EAs)

EAs are population-based metaheuristic optimization algorithms where their central concepts, e.g., reproduction, mutation, recombination and selection are inspired by mechanisms of biological evolution. A population is a set of individuals representing candidate solutions. These individuals are used during their evolution to explore many areas of the solution space at the same time. They are evaluated by a fitness function representing the objective function(s) to determine their suitability as solutions for the problem. The basic principles of biological evolution like crossover, mutation and selection are iteratively applied to the tentative solutions for generating new offsprings within each generation until a termination criterion is reached [187, 82, 154]. Generally, EAs have the following terminologies:

**Population** A population can be viewed as a finite set of possible solutions of a problem. The diversity of the population plays a key role in exploring the search space to find suitable solutions for the considered optimization problem. An initial set of candidate solutions in the form of an initial population is generated before starting the process of evolution. The population size is set in advance and typically fixed through the evolution steps. To guarantee a highly diverse population, several strategies and methods are applied. The most simple and straightforward method is to generate the initial population randomly. Other methods such as using prior solutions of a previous run of an EA as an initial population for the next run or combining the random method with prior solutions are other possibilities to generate a diverse initial population. The initial population is an essential part of the evolutionary process, since a population with a wide range of diverse individuals increases the opportunity of the EA to find a near-optimal solution or even the optimal one.

In case of an optimization problem with constraints, members of a population (i.e., individuals as candidate solutions) need to be feasible solutions, i.e., they typically satisfy the constraints. Even if all individuals of one population satisfy the constraints, genetic operations in general can produce new individuals for the next generation which can violate those constraints. Therefore, special strategies are needed for applying EAs to optimization problems with constraints (see later).

**Genetic Representation** An EA needs a representation schema for encoding the proposed solution in a genetic representation. In 1866, Mendel discovered that nature distinguishes the genetic code of an individual and its outward appearance (see, [140] for more details). I.e, each individual is a dual entity, represented by the internal genotypic properties and the external phenotypic properties where the genotype encodes the phenotype. Successful and effective use of an EA requires an appropriate internal representation of individuals. Several methods are proposed to encode the potential solutions, namely binary, characters, mixed representations, vectors and lists, to name a few. Since the representation of a solution is not predetermined by a method, a problem can be mapped to different kinds of encodings. This complicates the process of choosing the most proper coding schema which allows an efficient application of the genetic operators. A common representation scheme for individuals is described by a sequence (list or array, called a chromosome) of smaller building blocks (called allele) which are instances of pre-given structural schemas (called gene) for certain building blocks analogous to the structure of RNA/DNA. Figure 2.1 illustrates an exemplary population described by a character encoding of the gene information of an allele. Precisely, the special variation of a gene within the chromosome is described by distinguishing variations by unique character strings of an alphabet. The example population consists of three individuals described by three chromosomes where each one stores all genetic information of an individual. In contrast to nature, EAs use one chromosome to encode the genotypic information of an individual representing one possible solution candidate for a specific problem [12, 159]. Each chromosome is described as a sequence (list) of alleles, where each allele (or a list of alleles) is a special variation (instance) of a certain gene. In the illustration below the allele(s) of a gene are encoded as a character string. In nature, an allele exists in pairwise or more variations and represents the smallest information unit in a chromosome [12, 159, 87]. For encoding real-world problem settings into software-defined EA solutions, genes are often described by a schema definition of a more complex data structure having several attributes where the alleles are then concrete instances of this data structure.

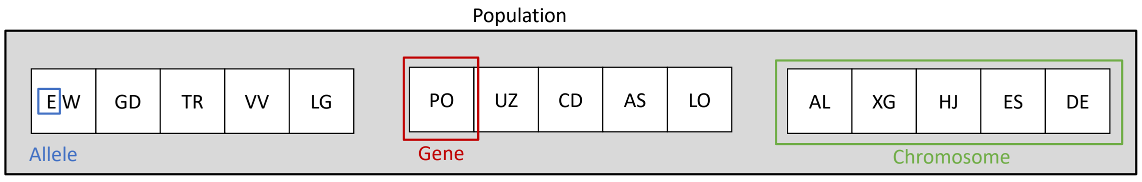


**Figure 2.1.:** Illustration of allele, gene, chromosome and population

**Fitness Function** The fitness value is a measure that determines the potential of an individual to provide a good approximate solution for the considered problem. The way of calculating this value depends on the concrete optimization problem and its real-world context. Typically, the fitness function for calculating the fitness value somehow makes use of the objective function(s) of the given optimization problem. E.g., the fitness value of an individual is higher if it better approximates the minimum criteria of a given single objective function. For multi-objective optimization problems having more than one objective function, the fitness function often has to weigh between the otherwise conflicting optimization goals of each single objective function. For solving optimization problems with constraints using EAs, one possible solution is to encode the constraints into the fitness function by defining penalty functions which drastically decrease the fitness of an individual if it violates constraints [108].

An EA evolves (modifies) a population by selecting and applying genetic operators to some or all of its individuals generating a new population until, e.g., the fitness of one or more individuals in the population reaches a fitness value which is defined as a termination criterion. Typically, some individuals with already good fitness will be carried over to the next population (the Offspring) unchanged which is called survival of the fittest [108].

**Genetic Operators** There are two main basic forms of genetic operators, namely recombination (e.g., crossover) and mutation. The crossover operation swaps the genes of the parents until a predefined crossover point is reached as shown in Figure 2.2. The new combined individuals are considered as children or offspring. This type of crossover is known as single-point crossover. Multi-point crossover is a similar operation of crossover, however, instead of having only a single crossover point, it has two or more ($n$ crossover points). Therefore, it is also called $n$-point crossover. Generally, both operations select the positions of the crossover points, randomly. Other forms of recombination operators are possible, where the operation somehow recombines the alleles of two or more individuals in other ways to create new child individuals. Note that such recombination operators require a kind of list representation of the individuals (i.e., a form of chromosome structure consisting of instances of genes) because they swap or exchange alleles at different places within the list. A mutation operator of an EA changes the value representation of the individual somehow but maintains its internal structure. E.g., if the structure is represented by a bit field, it could randomly flip one or more bits of the bit field but does not remove bits from the representation. As a genetic operator, a mutation typically only modifies the internal value representation of one or more alleles preserving different instances of the same corresponding gene(s). Therefore, the overall genetic structure of the individuals is maintained leading to another candidate solution (i.e., individual) for the given problem. By the application of different genetic operators to a set of parents and survival of the fittest, an offspring of a population is generated inheriting the genetic information from the parents but also containing some new variations of the chromosome structure. On one hand, this allows exploration of new parts of the search space by maintaining a diversity of the population and therefore decreasing the risk of premature convergence. On the other hand, it passes on the solution quality of the fittest parents for comparing their fitness with future offspring.

**Termination Criteria** Generally, there are two main termination criteria for EAs, namely the solution quality and the EA effort. By applying the solution quality criterion, an EA

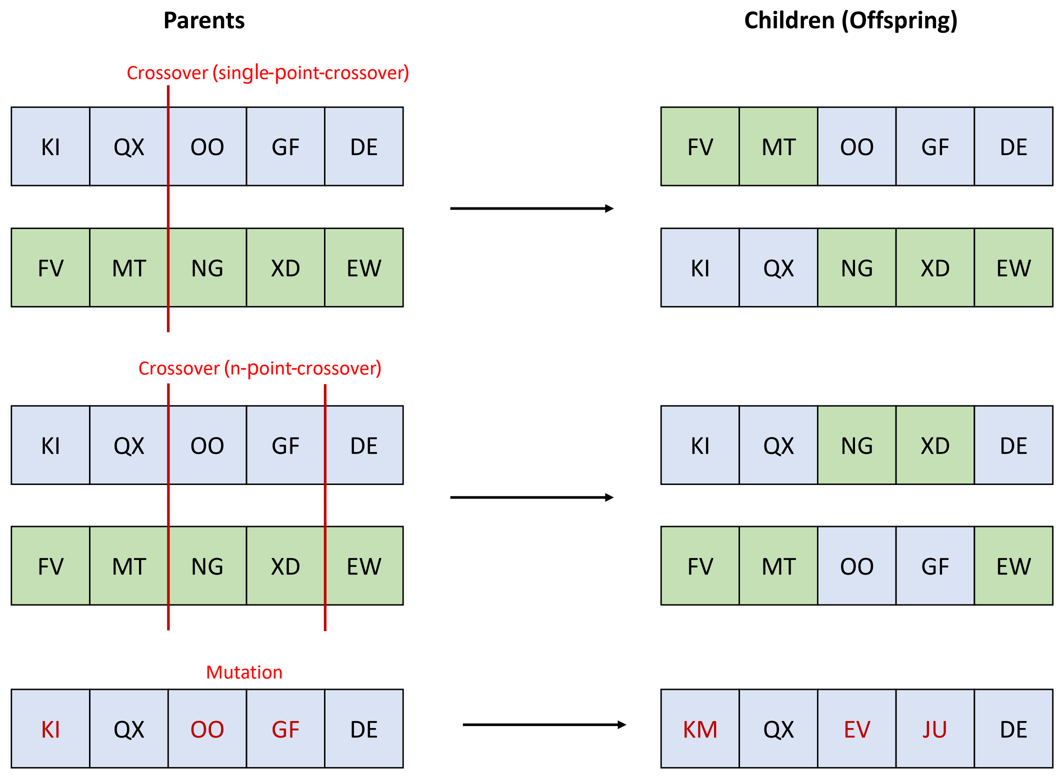


**Figure 2.2.:** Exemplary application of evolutionary operations on genes that consist of five character tuples

is stopped if one or multiple individuals reach a predefined fitness value. This termination criterion does not guarantee the termination of a running EA, since it cannot be guaranteed that the predefined fitness threshold value is reachable leading to an infinite runtime of the EA. Therefore, the EA effort criterion is defined by the number of generations after which the EA stops the evolution and therefore the EA process. Other realizations for the EA effort criterion such as predefined runtime for an EA or the number of evaluations performed during the evolution process can be applied. For both criteria and at the end of the evolution process, the EA returns the fittest individual as a solution for the considered optimization problem.

In summary, Figure 2.3 depicts the generic workflow of an EA where in the first step, an initial population is created. For generating a population with sufficient genetic diversity, it is preferred to generate 75-90% of the initial population randomly [88]. Moreover, other algorithms can be used to assist EAs by completely or partially creating the initial population of an EA based on domain-based knowledge, e.g., an initial population generated by a greedy algorithm [122]. In the second step, the initial population is evaluated to assign a fitness value to each individual. These values are necessary to select the parents of the next generation. As soon as all individuals are evaluated, the reproductive cycle starts. In the first step of a reproductive cycle, the parents of the next generation are selected. Afterward, crossover or more general recombination operators are applied to generate new individuals from the parents. In the mutation step, some properties of the newly generated individuals are randomly modified by randomly changing the genetic information, i.e., creating new alleles. At the end of a reproductive cycle, the new offspring is evaluated and then the new population is selected by combining the fittest parents with offspring candidates (according to the evaluation) for the next generation. Afterward, the next loop of the reproductive cycle is started. If one of the termination criteria is satisfied, the reproductive cycle is stopped and the EA outputs the best individuals as solutions.

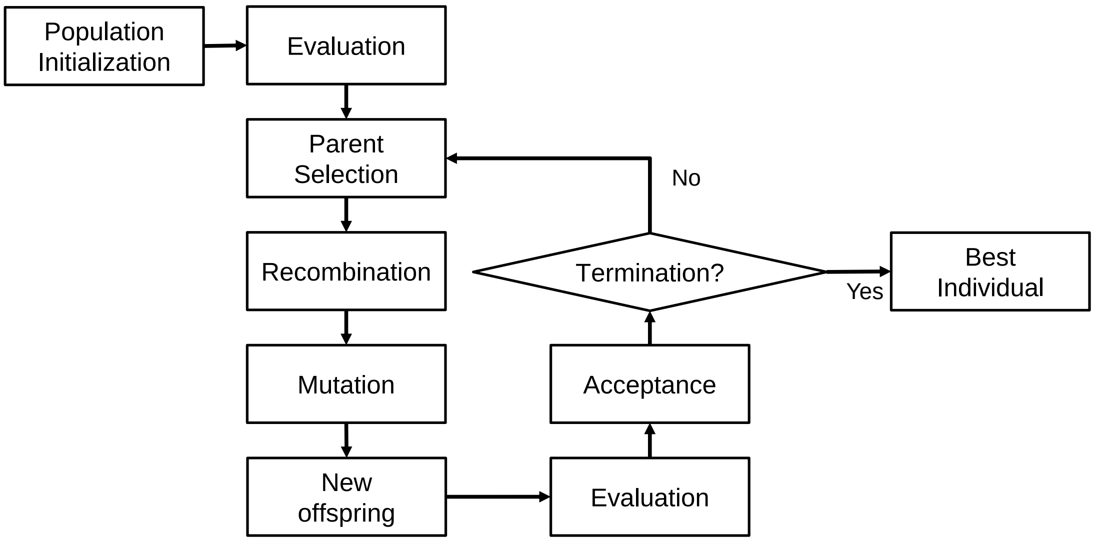


**Figure 2.3.:** The general scheme of an evolutionary algorithm as a flowchart

The most important aspects of an EA are the selection of parents, the acceptance of offspring, i.e., the replacement strategy and the application of the genetic operators. The application of these aspects is used to characterize and classify EAs into two categories, namely Panmictic EAs and Structured EAs [4]. In Panmictic EAs, the parent selection process is globally applied on the whole population. In other words, any individual can potentially mate with any other one in this population. In contrast to Panmictic EAs, Structured EAs have a spatial population structure restricting the application of operators to a subpopulation. Structured EAs divide the population into several subpopulations where the selection, replacement and generic operators are separately applied on each subpopulation. The main difference between Panmictic and Structured EAs is the decentralized selection and the replacement of individuals for the reproductive cycle. At first, Structured EAs seem to be more ‘natural’, because in nature mating can only happen when the individuals live in the same geographic area. But in biology, individuals of a species can travel over long distances, so that an individual can move from one subpopulation to another. This exchange of individuals between otherwise isolated subpopulations can also be integrated into Structured EAs, whereby an exchange of individuals can be done after each evolution step or only occasionally. This will become important in the next sections when we discuss the parallel models of EAs.

For the optimization problems where the solution has to strictly adhere to some constraints, a general problem of EAs is that there is no guarantee to generate offspring by applying the genetic operators without a violation of constraints, even if the parent individuals satisfy all constraints. Therefore, the evaluation step after generating the new offspring has also to decide how to deal with constraints. The acceptance process can completely ignore the constraints during the intermediate evaluations in the hope that the inclusion of the constraints in the fitness function, e.g., using penalty functions, would lead to a final solution with good fitness and adheres to the constraints. Otherwise, constraint violations need to be handled by the acceptance process, e.g., by either slightly modifying the generated childern so that they fulfill constraints or by removing children from the new offspring when they do not satisfy constraints. For the first option, a general procedure for casting individuals into feasible individuals satisfying the constraints is needed which does not decrease their fitness too much. By the second option, very small offspring can be resulted by removing individuals that do not commit to the considered constraints. In this case, the need to repeat the generation of offspring until the new population is large enough is not ideal either [171].

### 2.2.1. Parallel Models of EAs

Different classifications of parallelization models for EA are existing In the literature, e.g., [69, 117, 35, 122, 70]. The most common classification is the one introduced by Cantú-Paz [35], in which three basic parallelization approaches of EAs, namely the Global Model, the Coarse-Grained Model (Island Model) and the Fine-Grained Model are presented and investigated. On the one hand, if the parallelization is achieved only by distributing the evaluation step over several computing units, a Global Model is instrumented. On the other hand, if a spatial distribution of individuals, either in the form of neighbors or islands, is applied using a Structured EA and the application of the operators is limited to these subpopulations and thereby parallelized, a Fine-Grained or Coarse-Grained Model is applied, respectively. If the probability of 'maturing' (or more generally of applying operators) on sets of individuals is the same for all chosen subsets of a neighborhood or island, such subpopulations are also called 'Deme' analogously to biology.

**Global Model** The Global Model [35, 34, 5, 4] (so-called Master-Worker Model) depicted in Figure 2.4 distributes only the evaluation of the individuals to several workers. Each worker performs the evaluation on a distinct subset of the population. The Global Model does not change the behavior of the EA, since the evolutionary operators are only performed by the master on the parent selection as in Panmictic EAs. Therefore, the simplicity of the model is considered as a main advantage. The master stores the individuals, sends subsets of them to the workers to be evaluated and finally collects the results from the workers. If all workers try to communicate with the master at the same time, the master becomes a bottleneck. Consequently, the Global Model is only efficient when the computational costs of the fitness function are very high compared to the communication costs. The Global Model is called synchronous when the master waits for all workers to complete their evaluation before continuing. This leads to inefficient processor utilization, since some workers take more time than others. For increasing the processor utilization, some asynchronous implementations, e.g., [156] are proposed. In asynchronous Global Model, the master does not wait until all workers are finished to continue the evolution process. Obviously, any EA distributed according to the synchronous Global Model has the same numerical behavior as the underlying serial Panmictic EA. The Global Model can be implemented on a shared-memory as well as on a distributed-memory computer [34], since there is no assumption about the underlying computing architecture.

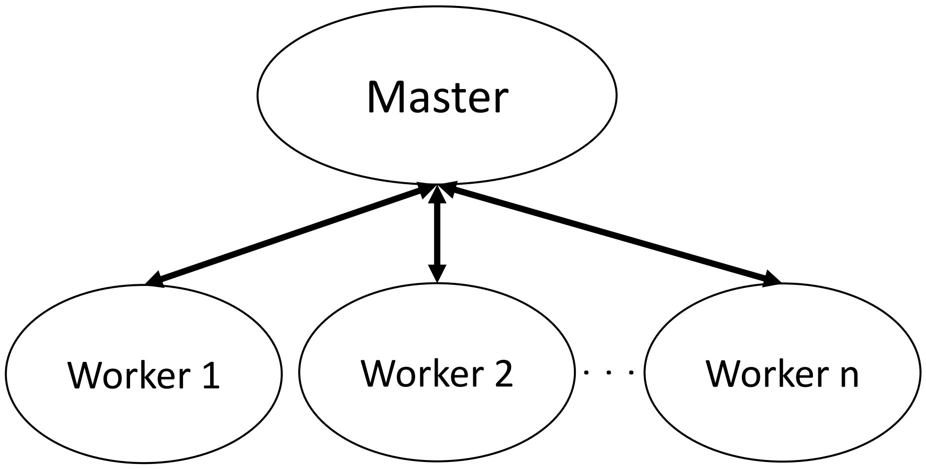


**Figure 2.4.:** Global Model adapted from [35]

**Coarse-Grained Model** In the Coarse-Grained Model [117, 122], the (initial) population is divided and spatially distributed into several subpopulations with a larger set of individuals called islands. Genetic operators such as recombination, mutation and replacement are only applied to the individuals of the same island and this can be done in parallel for each island, since the evolution in each island is performed independently. Additionally, the islands communicate periodically with each other according to a predefined communication topology to exchange some individuals (so-called migrants) as depicted in Figure 2.5. The communication topology determines how fast good solutions will disseminate to other islands influencing the performance of the Coarse-Grained Model. For example, dense connectivity results in a fast spread of migrants, while sparse connectivity slows it down. The number of generations between two succeeding migrations is defined by the migration frequency. The cooperation process among the islands assists the EA to maintain genetic diversity for a long time, reducing the problem of premature convergence and accelerating the search process. This is due to the fact that good solutions come together to form potentially better solutions [35]. Despite such advantages of the Coarse-Grained Model, this model exposes many parameters to be set and tuned for introducing an optimal performance making it more complex than the other parallelization models [122, 35]. Some parameters and their meaning are:

- **Migration rate:** It determines how many individuals will be exchanged between the islands in one migration process. A high migration rate can rapidly provide a solution with a quality similar to the one provided by sequential EAs as stated by Cantú-Paz in [35]. However, this leads to a high level of communications between islands. The amount of exchanged individuals can be determined either proportionally to the population size or by an absolute value.

- **Migration frequency:** It is the interval between two succeeding migrations (also called epoch [37]). It is usually determined by a local epoch termination criterion such as the number of generations or a predefined fitness value. In [68], it is proven that a high migration interval can significantly reduce the risk of premature convergence and communication overhead between the islands. However, this slows down the evolution speedup, since good solutions slowly spread between the islands.
- **Selection policy:** It decides which individuals of the current subpopulation are selected to be exchanged with the neighbors. Generally, the best individuals based on their fitness values are chosen as migrants. This helps the spread of good genetic information to other subpopulations and the acceleration of the evolution process. However, the random selection of migrants can also be applied.
- **Replacement policy:** It determines which individuals in the target subpopulation will be replaced by the migrants (if any). Usually, the worst individuals which have a low fitness value in the target population are replaced. Alternatively, a random replacement of individuals can be used.
- **Migration policy:** It defines how the islands migrate the individuals. There are two policies, namely synchronous or asynchronous. If a synchronous migration policy is applied, all islands migrate individuals at the same time, i.e., they wait for each other to finish the evolution. However, by applying an asynchronous migration policy, each island performs the migration process as soon as the migrants are ready and then continues with the next epoch.
- **Island homogeneity:** Generally, there are two types of homogeneity, namely homogeneous and heterogeneous. While in a homogeneous model, all islands have the same configurations such as EA configuration, in a heterogeneous model each island has its own configuration providing great flexibility to adapt settings on each node in a distributed system. This provides the ability to better keep genotypic diversity in the populations.
- **Communication topology:** It plays a key role in the Coarse-Grained Model, since it defines the neighbors of each island and between which islands the migration can occur. A dense topology ensures that the good individuals have a big chance to spread quickly between many islands [35] increasing the performance of EA. In [38], Cantú-Paz et al. show that there is an optimal number of neighbors where the Coarse-Grained Model introduces the best performance. If the number of neighbors is far below this number, the Coarse-Grained Model does not show a significant parallel speedup. Moreover, if the number of neighbors is far above this number, the communication overhead outweighs the gained speedup.

**Fine-Grained Model** In the Fine-Grained Model [122, 107], the population is spatially structured according to an underlying structuring scheme, such as a grid or lattice. Each individual is a member of one or more subpopulations (demes) called neighborhoods of the same size and structure defined by, e.g., a distance operator in the grid or lattice (see, Figure 2.6). Typically, neighborhoods will overlap and the application of the genetic operators

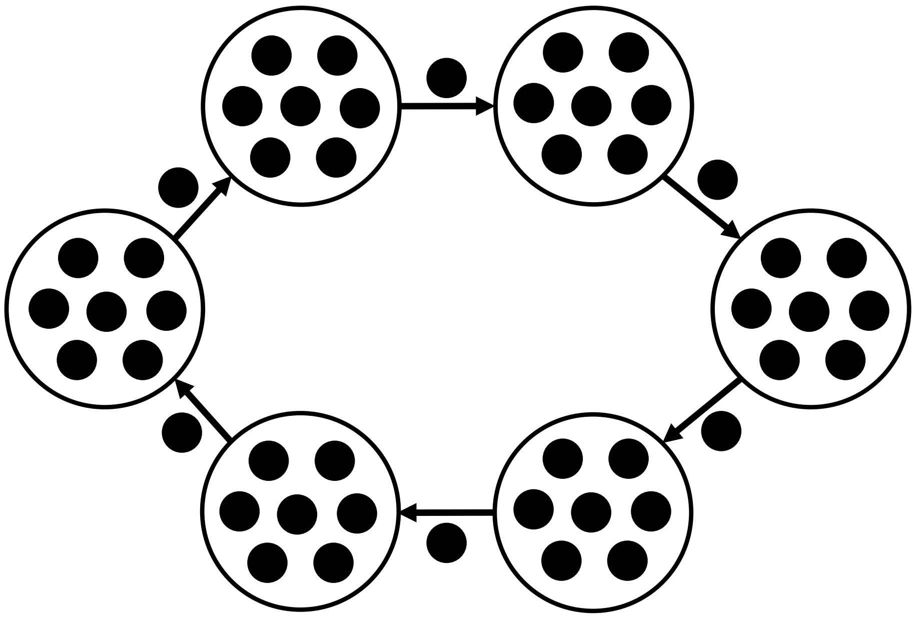


**Figure 2.5.:** Coarse-Grained Model with ring topology adapted from [35]

is then restricted to the neighbors, i.e., an individual can only mate with neighbors of the same deme as shown in Figure 2.6. Despite this restriction, genetic information can be spread due to the overlapping of the demes. As demes are independent of each other, genetic operators on different demes can be executed in parallel. By adjusting the selection pressure and the deme size, it can be controlled how fast the genetic information spreads across the population, preventing premature convergence. In other words, the ratio between a depth-first (exploitation) and breadth-first search (exploration) can be regulated by choosing those two parameters. In contrast to the Island Model, on each computing, node the evolutionary operators are only applied to a small set of individuals (e.g., a neighborhood) but due to the overlapping nature of neighborhoods, the changes have to be then propagated to many other computing nodes. The main advantages of the Fine-Grained Model are that it has significantly few strategy parameters to be tuned and applying genetic operators can be performed much quicker. However, this is offset by the disadvantage of a lot of communication which occurs especially at the beginning of an evolution run.

**Hierarchical Model and Non-Classical Models** The combination of two or more models from the above-mentioned EA parallelization models leads to a Hierarchical Model (so-called Hybrid Model) [122, 4, 5, 6, 34]. A Hierarchical Model is composed of multiple layers and usually implements a Coarse-Grained Model in the upper layer and one of the three basic models in the lower layer as shown in Figure 2.7 where the Coarse-Grained Model is combined with the Fine-Grained Model. The major advantage of the Hierarchical Model lies in limiting the disadvantages of using a single parallelization model alone while keeping the respective advantages of each model. Moreover, it provides great flexibility to design each layer separately.

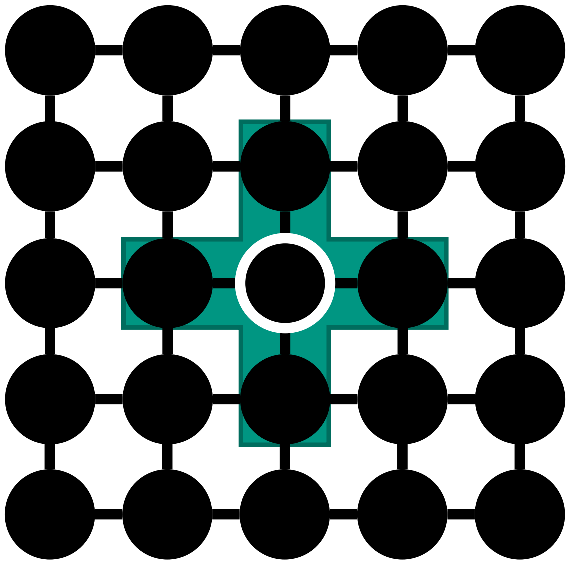


**Figure 2.6.:** Fine-Grained Model with grid structure adapted from [35]

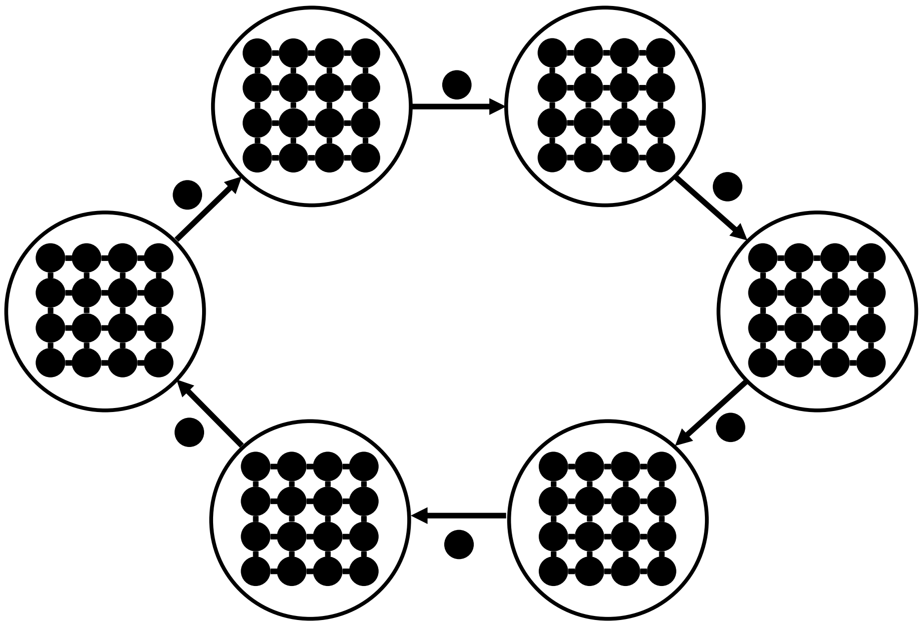


**Figure 2.7.:** Example of a Hierarchical Model combined the Coarse-Grained Model with the Fine-Grained Model adapted from [35]

In addition to the above-mentioned basic models, Gong et al. in [68] explain in their survey the Pool Model, the Coevolution Model and the Multi-Agent Model as different implementation strategies for parallelized EAs. The Pool Model is highly influenced by an underlying massive parallel computation platform where a set of autonomous processors is working on a shared resource pool. The processors in parallel execute the evolution of neighborhoods while the individuals are located in a shared resource pool, e.g., distributed object or memory cache. Moreover, the coupling is very low between them by defining only small neighborhoods, introducing the Pool Model as a highly parallel solution for parallelizing EAs. In contrast, the Coevolution Model and the Multi-Agent Model decompose the problem into smaller sub-problems by distributing the dimensionalities of the problem into several dimensions. Indeed, we focus in the present work only on the classic models and Hierarchical Model.

### 2.2.2. Hybridization of EAs

One popular method to enhance the performance of EAs and overcome their limitations is to hybridize them with other algorithms and methodologies for performing parts of the EA workflow. For example, utilizing cellular automata, neural networks, heuristics, etc., to seed the initial population, define the survivor selection, approximate the fitness function or even replace whole genetic operations. This enables EAs to successfully handle complex real-world optimization problems which involve imprecision, uncertainty, noisy environment and vagueness.

Grosan and Abraham [72] identified the most used hybrid architectures for EAs as follows:

- Hybridization between two or more EAs
- Fuzzy logic assisted EA
- Particle Swarm Optimization (PSO) assisted EA
- Ant Colony Optimization ACO assisted EA
- Bacterial Foraging Optimization assisted EA
- Hybridization between an EA and other heuristics
- Hybridization between an EA and Machine Learning (ML) techniques

Due to the fact that all core elements of EAs can be altered, numerous methods and techniques to investigate the above-listed hybridization approaches are proposed. For example and as shown in Figure 2.8, the hybridization approach can be applied for generating the initial population, estimating the fitness function and applying the genetic operators and the selection policies [173, 190, 167, 118, 64]. The diversity of the initial population plays an essential role in the search process of EAs. Therefore, enriching the initial population with good and diverse individuals improves the overall performance of EAs. To this end, other heuristics methods and ML techniques, namely Local Search (LS), Simulated Annealing (SA) and Artificial Neural Networks (ANN) are combined with EAs to assist them by generating a more diverse initial population (see, [72] for more details). For example, the initial population is seeded with the prior found solutions of a similar problem where each seed serves as a pointer to a promising area in the solution space assisting EAs to reach the optimal solution or the near-optimal one in a reasonable time. For complex and large-scale optimization problems, EAs require large numbers of fitness evaluations which are in most cases computationally expensive [76, 75, 186, 124, 15]. For accelerating the evaluation process, estimating the fitness value of each individual by instrumenting ML techniques like ANN or clustering algorithm is a feasible approach. Furthermore, tuning EA configuration like the population size, probability of mutation, crossover and recombination by incorporating domain-specific knowledge into EAs or combining EAs with the previously mentioned methods and techniques can generally enhance the search process of EAs.

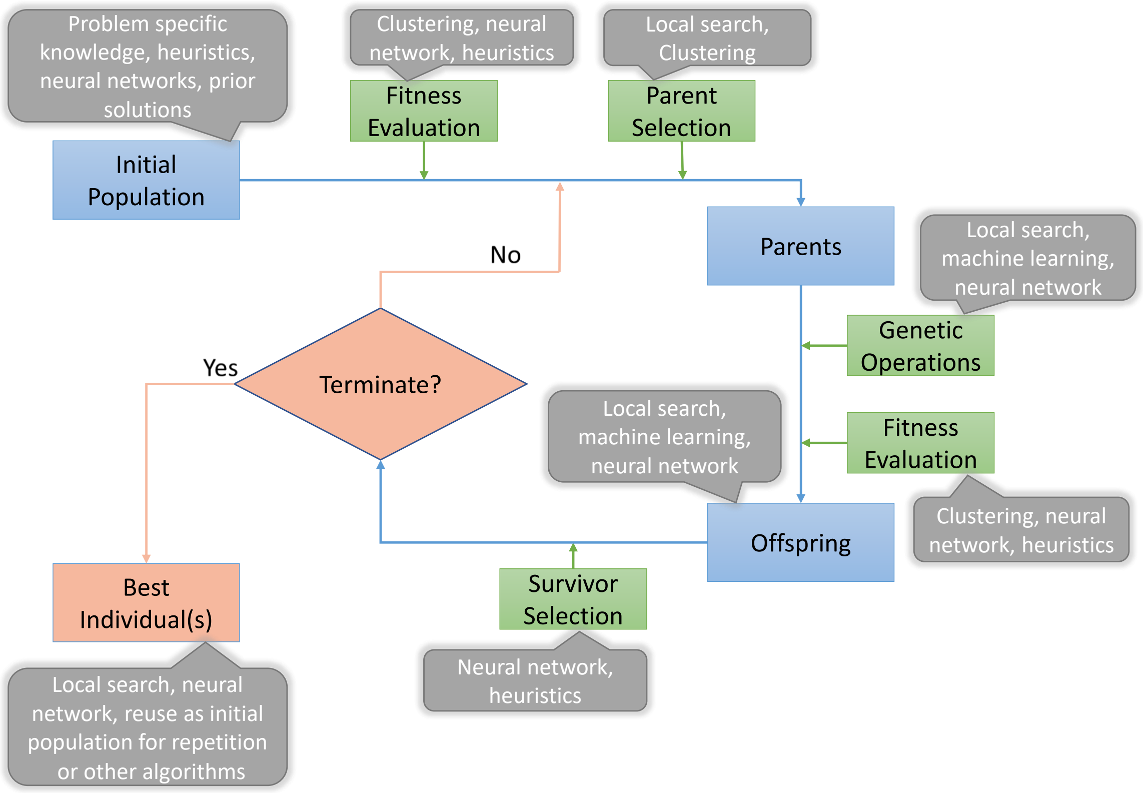

Problem specific knowledge, heuristics, neural networks, prior solutions
Clustering, neural network, heuristics
Local search, Clustering
Fitness Evaluation
Parent Selection
Initial Population
No
Parents
Local search, machine learning, neural network
Genetic Operations
Yes
Terminate?
Local search, machine learning, neural network
Fitness Evaluation
Clustering, neural network, heuristics
Offspring
Best Individual(s)
Survivor Selection
Neural network, heuristics
Local search, neural network, reuse as initial population for repetition or other algorithms

**Figure 2.8.:** Hybridization perspectives for an EA

### 2.2.3. The EA GLEAM

The EA GLEAM [1] (General Learning Evolutionary Algorithm and Method) [24, 23] is used in this work as an implemented example of an EA with great flexibility to evaluate the proposed method for facilitating the usage of EAs in cluster computing environments. GLEAM is originally designed to plan, optimize and control dynamic processes like collision-free path planning for industrial robots. GLEAM is distinguished from other EAs by its flexible coding, in which the decision variables that semantically belong to one gene are grouped together. Each gene defines a type schema that determines how many decision variables of which data type with which lower and upper limits it contains. A decision variable can either be of type Integer or Real. The upper and lower limits restrict the range of the Integer or Real variable to a certain interval with lower and upper boundaries and can be seen as a simple form of inequality constraint conditions. In GLEAM, such a gene definition (also called action type) is always identified by its type ID (a unique Integer). The limits of the decision variables are statically assigned to the gene and cannot therefore be changed by evolution, but dynamically changeable limits can also be modeled as decision variables if needed. The set of gene definitions defines the genotype and therefore the semantic structure of the alleles for a given problem setting.

[1] https://github.com/KIT-IAI/Gleam (Accessed: 10.02.2023)

For example, for scheduling power generators or storages to fulfill a given demand, the EA GLEAM defines a flexible type schema to code the so-called scheduling operations in genes, in which the type ID represents the unit ID for each power generation unit. Each gene consists of, e.g., start time and duration as Integer decision variables as shown in Figure 2.9a. If some of the power generation units are operated at a certain fraction, e.g., power storages, a new decision variable is added to the genes, namely power fraction which takes Real values as shown in Figure 2.9b. For ensuring that GLEAM does not generate infeasible values for these decision variables, an appropriate value range depending on the studied optimization use case is defined for each decision variable. (cf. [86, 23, 24]) for a detailed discussion). Figure 2.9 shows an example of such limits where the optimization problem is the creation of an hourly day-ahead schedule plan for a microgrid consisting of 25 power generation units.

| Type ID (Unit ID) [1,25] | | |
|---|---|---|
| **Decision Variable** | **Data Type** | **Limits** |
| start time | Integer | [0,24] |
| duration | Integer | [0,24] |

**(a)**

| Type ID (Unit ID) [1,25] | | |
|---|---|---|
| **Decision Variable** | **Data Type** | **Limits** |
| start time | Integer | [0,24] |
| duration | Integer | [0,24] |
| power fraction | Real | [0,100] |

**(b)**

**Figure 2.9.:** An example of a type schema proposed by GLEAM for scheduling building 24-hours day-ahead schedule for 25 different units of power generation, adapted from [90]

The chromosomes contain one or more instances of those genes that are described by their type definitions as a list of actions (i.e., gene instances representing alleles). The actions are additionally grouped together into one or more segments, whereby the segment boundaries can be dynamically changed or set randomly according to some knowledge about the application (see, Figure 2.10). Genetic operations, e.g., mutation and crossover can then be applied to the whole segments. This will allow the evolution of partial solutions as a whole.

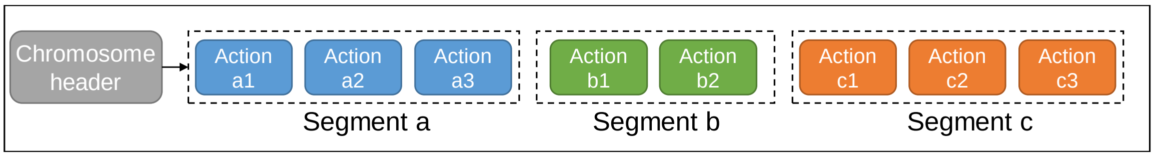


**Figure 2.10.:** Example of a list of actions with segments forming an individual, adapted from [24]

GLEAM currently supports three basic sub-types of chromosomes. In its simplest form, every gene type is represented by exactly one gene instance in the chromosome and the order of the instances does not matter. Secondly, the chromosome contains only one instance of each gene but the order matters. Thirdly, the chromosome can contain more than one instance of each gene and the order is relevant. The first subtype is appropriate for static optimization problems where the structure of, e.g., a technical system and its dynamics does not change and the application needs only to find the best set of parameters for configuring the system according to an underlying optimization problem. The second subtype is especially useful for combinatorial tasks or tasks where both the sequence of the elements and their parameters are of interest. The third subtype is the most appropriate for optimizing dynamic processes where even the technical system can be dynamically changed, e.g., by sending control signals to system parts for modifying the behavior of the system. E.g., sending control signals will result in having an action element performing control on a certain machine several times in a chromosome [23, 24, 90].

The described gene types and the rules for chromosome construction and manipulation form the so-called gene model which defines the GLEAM configuration for performing a set of general genetic operators on populations. GLEAM provides a set of basic genetic operators which can be classified into segment-specific (e.g., add or delete segments or move their boundaries), action-specific (e.g., add, delete or duplicate actions or mutate their parameters), mutation of parameters (e.g., smaller or larger changes of parameter values) and Crossover (one-point, n-point and segment crossover) operations. GLEAM additionally implements a specific approach for applying some of these operators in a specific order by grouping a number of them into a self-defined evolutionary operator. Each basic operator also has an execution probability which defines if the basic operator is actually applied when the self-defined evolutionary operator is executed. Moreover, this approach provides the ability to add problem-specific genetic operators to the set of general ones and therefore to use own defined evolutionary operators (cf. [23, 24], for a detailed discussion).

Note, that the exact set of operations applied depends on the specific subtype of chromosome model. For instance, crossover can now operate on segments as a whole or on single actions and mutations can be applied to specific attributes of the gene type (decision variables) and can respect their data type and boundaries. Hence, explicit restrictions that are common in real-world problems can be easily and advantageously handled in the gene model of GLEAM. The violation of other more implicit constraints is detected in GLEAM during the evaluation of a chromosome. This violation is then quantitatively determined and implemented as a penalty function which provides a penalty factor between zero and one.

GLEAM provides the ability to evaluate the generated chromosomes either externally by sending it to an external service – this facilitates the parallelization of GLEAM according to the Global Model – or internally. By default, GLEAM generates the initial population completely randomly. However, it supports the integration of some individuals which are taken from previous runs of GLEAM for solving the same or similar optimization problems or generated by an external tool or algorithms into the initial population. This feature enables the parallelization of GLEAM based on the Coarse-Grained Model and the hybridization of GLEAM with other algorithms for seeding the initial population. The population structure of GLEAM is based on the Fine-Grained Model with overlapping neighbors. The individuals are organized as a ring in order to find the right balance between width- and depth-first search.

Since we cannot assign each individual of a population to a processor due to the limitation of availability of CPUs in the used computing cluster, we do not consider the Fine-Grained Model of GLEAM in this work. GLEAM is written in C and the interaction between it and the implemented services is realized in terms of invocations based on the command-line calls. Data exchange is accomplished through file access.

## 2.3. Parallel Computing

In parallel computing, complex calculation problems are broken down into several small calculations which are carried out at the same time by a distributed system [10]. This offers great opportunities regarding the performance of the system and other beneficial traits. According to Coulouris et al. [42] and Tanenbaum et al. [179], the distributed systems can be defined as systems that consist of components such as computers, printers or running computer programs connected by a network for exchanging messages among them. It is very important to the end users to see a distributed system as a single coherent system. The components of a distributed system are connected and related to each other according to a predefined topology which has a great influence on the characteristics of distributed systems [179].

The communication between the different components of a distributed system is very essential. There are three types of communication paradigms, namely inter-process communication, remote invocation and indirect communication.

- Inter-process communication refers to a low-level communication between the processes in the distributed systems. This includes message-passing primitives, two-way streams and multicast communication.
- Remote invocation is a two-way exchange between connected components in a distributed system. The benefits include calling procedures on remote components as if they are local and better support for object-oriented programming.
- Indirect communication is defined as the communication between components in a system is performed through an intermediate layer. Therefore, there is no direct coupling between the sender(s) and the receiver(s). This makes the communication more flexible in terms of topology change and increases the flexibility of adding new components.

Remote invocation and indirect communication are the two communication types that are used in the context of this thesis. While the response-request communication paradigm is used for the remote invocation type, the event-based communication paradigm is utilized to realize the indirect communication type. The response-request communication paradigm describes how two services can directly communicate with each other where one service initiates a request to another and in return expects a response. The event-based communication paradigm is driven by events where one component (so-called producer) emits an event and all components that have subscribed to the event type (so-called subscribers) will get an update. REST (REpresentational State Transfer) is a protocol-agnostic architectural style that does not rely on a specific protocol. Generally, REST implements the response-request communication paradigm by using the Hypertext Transfer Protocol (HTTP) as an underlying communication protocol. HTTP is a stateless application-level response-request protocol for the foundation of data communication on the web. It is used for other interactions on the web as service-to-service communications. HTTP has five main methods that simplify the connection to the components and the manipulation of data. The GET method is used to retrieve/fetch data from a source. The POST method is used to create new resources. The PUT method replaces/updates an existing resource. The PATCH method partially updates an existing resource. The DELETE method is used to delete an existing resource. The main advantage of REST is based on its simplicity where many web applications support REST interfaces, since it provides less overhead and easy-to-handle data resource formats like JSON, HTML or plain text.

The publish/subscribe pattern is a common way to implement the event-based communication paradigm. The main goal of the publish/subscribe pattern is to loosely coupling senders and receivers reflecting the dynamic and decoupled nature of distributed systems. Instead of directly interacting between senders and receivers, the publish/subscribe messaging paradigm arranges events in channels. The sender (publisher) is programmed to send its events to one of the channels without knowing the receivers. The receiver (subscriber) in contrast only expresses interest in one or more channels to receive messages that satisfy its interest. The intermediate channel matches subscriptions against published events. Since there are potentially many subscribers for one channel, the publish/subscribe paradigm is essentially a one-to-many communication paradigm. This decouples senders and receivers in time, space and synchronization [51] allowing dynamic network topologies and many different configurations. Hence, the publish/subscribe messaging paradigm is well suited for the communication between several components allowing experiments with several changing configurations.

### 2.3.1. Service-Oriented Architecture and Microservices Architecture

The Service-Oriented Architecture (SOA) is an approach to break down the large monolithic applications into distinct several software blocks (so-called services). These services interact with each other to share their functionalities. In other words, SOA builds a distributed system with several software components where each one interacts with others through messages exchanged over a network rather than within a process boundary. This introduces SOA as a software solution to handle the changes and the heterogeneity that can be found in large monolithic applications. The major objective of SOA is the reusability where the services can be reused by two or more end-user applications or other services. However, SOA has several challenges like the lack of guidance for service granularity, communication protocol and wrong guidance to split the system [139], to name a few. To face such problems, microservice architecture has emerged from real-world applications. The microservice architecture is a specific approach of SOA which is composed of several independent deployable services [139]. Microservices are small, autonomous services that work together to perform a specific task. Since all services are independent of each other, each microservice can utilize its own technology stack allowing great flexibility, e.g., one of the deployed services is written in Python and the others in Java. The independence of the services also allows each service to scale on demand. The principles to design microservices to assure their previously mentioned characteristics are summarized by Newman [139] as the following:

- Model around business concepts: the interfaces are modeled around business bounds and are therefore more stable than those ones modeled around technical concepts.
- Culture of automation: the automation is used to address the problem introduced by further complexity because of many moving and changing parts.
- Hide internal implementation details: it ensures the independent evolution and the development of all services and their databases.
- Decentralize all the things: it maximizes the autonomy of all microservices and decision making. Especially, the decision to deploy services is made by the team that develops the services.
- Independently deployable: the services are closely related to the previous point. Here, the services and their different versions should be independently deployable and able to coexist.
- Isolate failure: the services should avoid cascading failures to other services. Therefore, a service should be able to handle failures occurring when calling other services.
- Highly observable: since a distributed system designed according to the microservice architecture consists of potentially many services and machines, observing the behavior becomes tedious. Therefore, it should be possible to observe the state of the whole system from a single point of view.

## 2.4. Machine Learning (ML)

Machine Learning (ML) is the process of a computer system learning patterns from data to make accurate predictions [135]. These predictions could be answered, e.g., whether an animal in a photo is a cat or a dog and whether an email is a spam or a normal email. Furthermore, ML can be used for more complex problems such as recognizing tempo limits on a road sign by a self-driving car. ML is often categorized by how a model learns to improve its prediction accuracy. The different basic approaches can be broadly grouped under four learning techniques [135]:

- Supervised learning requires labeled data, i.e., inputs known as features corresponding to their outputs called labels. Given sufficient training examples, a supervised learning model learns the mapping between the input and the output to make the predictions based on the learned knowledge [141].
- Unsupervised learning is a type of ML, in which an algorithm is trained on unlabeled data. In contrast to supervised learning, the algorithm identifies patterns in the data and potentially finds correlations that, e.g., describe the data to split it into similar categories [106].
- Semi-supervised learning is an approach of ML involving a mix of both above-introduced learning methods. The model is initially fed with a small amount of labeled data and afterward with a large amount of unlabelled data. The model is partially

trained with the labeled training data to label the unlabeled data. Finally, it is trained on a mix of labeled and pseudo-labeled data [39].

- Reinforcement learning [96] is a learning type, in which the model learns from the environment, since there is no available data to train it. The training of such a model is a multi-step process where the model makes a series of decisions which affect the state of this environment. If the decision is right, the model receives a reward otherwise it receives punishment. An example of such learning is autonomous driving such as Tesla.

In this thesis, the main focus is to use the first two learning approaches, namely the supervised and the unsupervised learning. While clustering techniques and Autoencoder are used for the unsupervised learning approach, Artificial Neural Networks (ANNs) are selected as an example of the supervised learning approach.

### 2.4.1. Clustering

Clustering is a density estimation method used to group similar objects into several groups or clusters [83]. There are diverse applications of clustering algorithms like document clustering, noise detection and image compression, to name a few. For example, in the energy field, customers that have similar consumption profiles can be grouped together into one cluster opening new perspectives for providing better services. To build groups of similar inputs, there are several clustering models [134, 78], e.g. distribution, group, centroid, density and connectivity models. In this thesis, centroid, density and connectivity models are used. While the centroid and connectivity models depend on the distance metrics such as Euclidean distance, Manhattan distance, Minkowski distance and Cosine similarity to define the similarity or dissimilarity between the input data points, the density model – as its name implies – considers the density of the input data to define the clusters. The most prominent implementation of the centroid-based clustering model is the KMeans algorithm [83]. The density-based clustering model is the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm and the connectivity-based clustering model is the Agglomerative Clustering algorithm. Besides these three algorithms, Affinity Propagation clustering algorithm is tested. K-Nearest-Neighbor (KNN) algorithm can either be used as a supervised or an unsupervised method. In the context of this thesis, it is used as an unsupervised approach to find the k nearest neighbors of a given object. Generally, these algorithms work better with low dimensional data [85], i.e., the higher the number of dimensions, the less accurate is the clustering.

### 2.4.2. Artificial Neural Networks (ANNs)

A neural network [109] is inspired by the human brain and consists of many simple processing units, so-called neurons which are illustrated in Figure 2.11. Each neuron of ANN has a mathematical function called the activation function which learns the relation between the input and the output. The most familiar used activation functions are the Rectified Linear

Unit (ReLU), Hyperbolic tangent (tanh), Softmax and Sigmoid. Each neural network has three layers, namely the input layer, the hidden layers (at least one) and the output layer as shown in Figure 2.11. The number of neurons in the first layer, i.e., the input layer is equal to the number of features in the training set. However, the number of neurons in the second layer, i.e., the hidden layer is different and there is no fixed rule to select this number. In most cases, the number of neurons in the hidden layer is smaller than in the input layer. The number of neurons in the last layer, i.e., output layer depends on the type of the problem. For example, for complex problems with multiple-output, ANN uses multiple neurons in the last layer equal to the number of outputs. This enables ANN to simultaneously predict multiple outputs and learn disjoint label combinations.

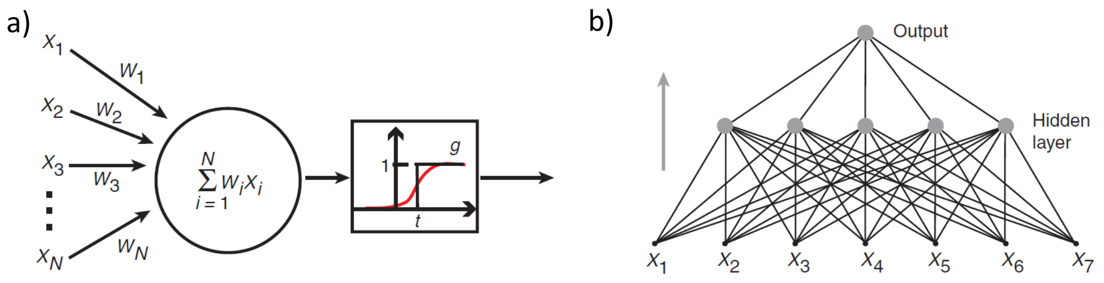


**Figure 2.11.:** Example of Artificial Neural Network a) Graphical representation of the McCulloch-Pitts model neuron threshold unit. b) Exemplary Feedforward Network [109]

**Autoencoder** An autoencoder is an unsupervised learning algorithm that leverages ANNs for feature reduction, e.g., [19] and anomaly detection, e.g., [40]. An essential characteristic of autoencoders is their lower dimension representation imposed by a bottleneck which is depicted, i.e., the Encoded Data layer in Figure 2.12.

The colors in the figure indicate similar representations of the data by the encoding stage and the decoding one. For feature reduction, the main task of an autoencoder is to map the input data from the original representation to another one with losing as little information as possible. In other words, the main aim of an autoencoder is to minimize the reconstruction error which measures the difference between the original input and the consequent reconstruction. In the encoder stage, the autoencoder is forced to find a lower-dimensional representation of the given data and simultaneously try to keep as much as possible the variance of the original data. In the decoder stage, the autoencoder reconstructs the input from the reduced features – the red layer in Figure 2.12 –. If the input features are independent of each other, this encoding and subsequent reconstruction are very difficult or even not realizable. However, if there is some sort of correlation between the input features, the autoencoder can eliminate them and successfully reduce the input dimensions [147].

[2] kvitajakub.github.io/2016/04/14/rnn-diagrams (Accessed: 10.02.2023)

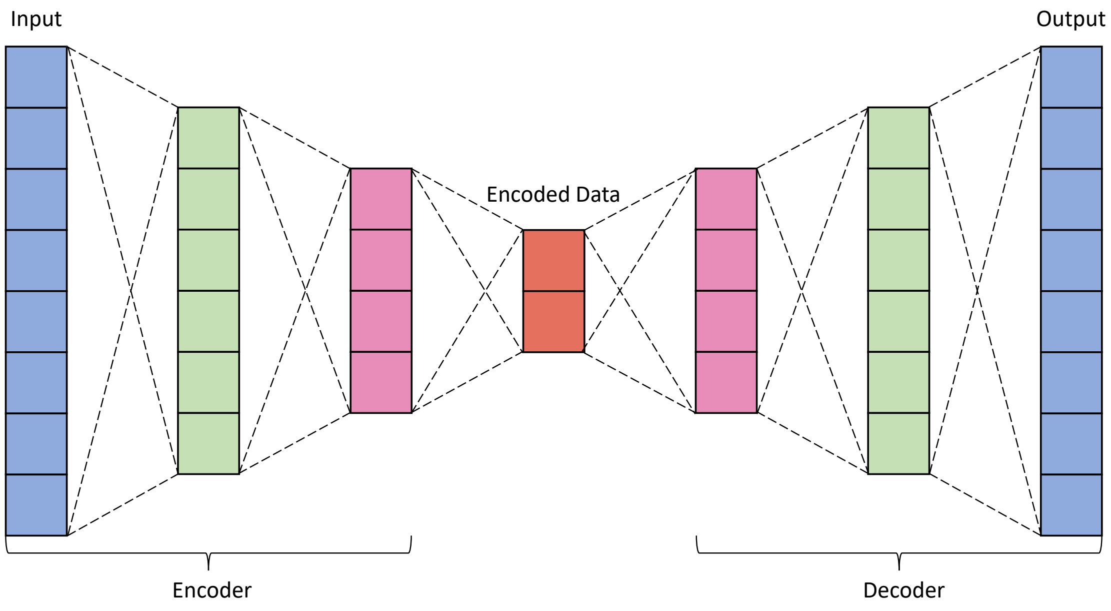


**Figure 2.12.:** Exemplary schema of a deep autoencoder with seven layers

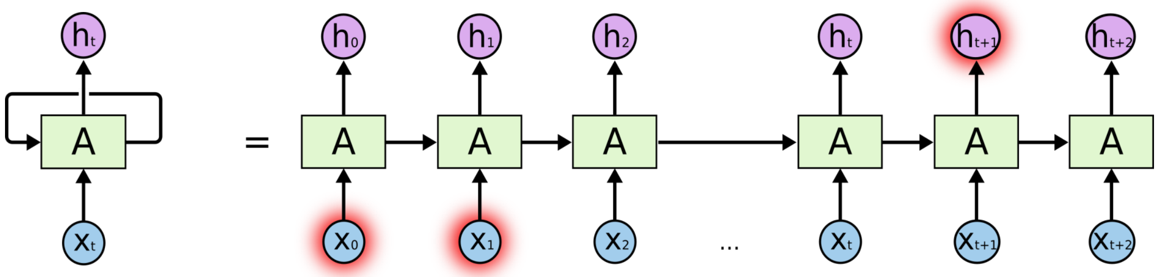


**Figure 2.13.:** Recurrent Neural Networks unrolled over time taken from [2]

**Recurrent Neural Network** In recent years, the number of applications that generate sequential data like speech recognition, language translation and the stock market has increased. Building a ML model that can learn from such data is considered as one of the most challenging problems in the ML field. Recurrent Neural Networks (RNNs) are a special architecture of ANNs that are mainly used to tackle this learning problem [129].

As shown in Figure 2.13, RNN receives input $x$ at time step $t$, outputs a value $h$ for the same time step $t$ and passes a message to the next time step $t + 1$. This mechanism enables RNN to work better if the information is saved in a specific sequence over time steps. It is important to note that RNN works better if the gap between the relevant information and the required information is small. In other words, RNN can remember the learned information for a short time. For example, when RNN predicts the last word of the sentence, 'the color of the ocean is . . . ', RNN can quickly predict the word without any additional information, since it is quite clear that the ocean is most likely blue. However, it is more difficult for RNN to predict the last word of the sentence, 'I come from . . . ' if multiple sentences such as ‘I study computer science in Karlsruhe which is a nice city located in the west south of Germany, . . .’ are coming before this sentence. The difficulty arises from the fact that RNN fails to remember what it is learned from long sequences [20]. For treating the long-term dependency issues, Hochreiter et al. in [80] have proposed Long Short-Term Memory Networks (LSTMs) architecture which has a similar structure of standard RNN. It consists of a simple and straightforward structure as depicted in the top part of Figure 2.14 where each cell is a single tanh layer. The same chain structure is used to build LSTM, however, supplementary operations besides the tanh layer are integrated within each cell. These operations are used to enable the LSTM to forget irrelevant information and keep the relevant one. The cell states and the three different gates are the main concept behind LSTMs. The cell states save and deliver relevant information to the next cell of the chain, ensuring that information is transferred from previous time steps to the next ones and reducing the effects of short-term memory. At each time step, information is added or removed from the cell state via the input gate and forget gate, respectively.

Sequence-to-Sequence problems (Seq2Seq) problems like the translation of sentences from German into English or the generation of captions for videos [182] are usually solved by applying LSTMs. In the simplest form, i.e., by translating word by word, one word is used as input for the LSTM model and the model outputs the corresponding translated word. Generally, the input and the output have different lengths leading to a special type of LSTM model which is called Seq2Seq LSTM model with multiple outputs. Such a model is typically built out of two main stages, namely an encoder stage and a decoder one. The encoder of the LSTM model processes the complete input sequence and returns its internal states discarding other outputs. These states can be interpreted as context and are used later as input for the decoder. In the first step of the decoder stage, the decoder gets the internal states of the encoder and a start vector as inputs to predict the first output (so-called target value). Then, it updates the states and passes them to the next time step. The decoder uses the target value and states from the previous time step to make the next prediction. This iteration is continued until the decoder generates an end vector as an indication of the end of the output sequence.

## 2.5. Technology Stack

In this section, the main software tools and technologies that are utilized for the investigations of this thesis are explained. Firstly, an introduction to the container-based virtualization approach is introduced. Secondly, an overview of the used data exchange and storage tools is given.

**Container-based Virtualization** Container technologies and Virtual Machines (VMs) are two of the most used approaches that allow the abstraction of the systems code from

[3] colah.github.io/posts/2015-08-Understanding-LSTMs (Accessed: 10.02.2023)

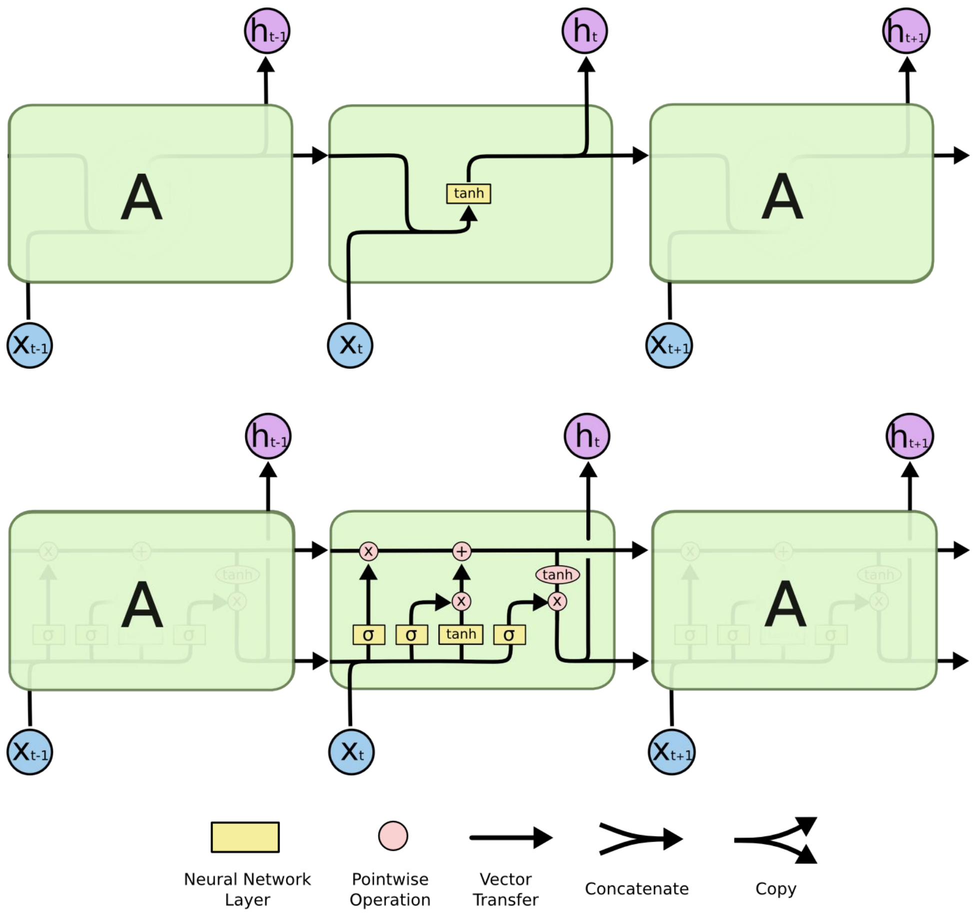


**Figure 2.14.:** The repeating module in standard RNN with a single layer is compared to the repeating module on an LSTM containing four interacting layers taken from [3]

the underlying hardware, simplifying automation, enabling portability and standardizing development across multiple machines and platforms. Container-based virtualization, also known as containerization, is an isolated virtual environment installed on top of an operating system kernel. The kernel isolates the environment for all running containers. As a result, there is no need to include an entire guest operating system in a container. Containers share the same properties regarding resource isolation and allocation with other virtualization techniques, such as VMs as shown in Figure 2.15. However, containers virtualize the operating system by enabling the run of isolated systems on a single server or host, i.e., the system resources are shared between all deployments. This makes the execution of applications running in containers highly efficient [49, 172], since one can scale up and down the computation power based on the current demand by increasing or decreasing the number of running containers. The most popular open source software enabling containerization is Docker [178]. Docker performs operating-system-level virtualization to isolate the applications. This is achieved by running containers on the Docker engine that separates the

[4] docker.com/resources/what-container (Accessed: 10.02.2023)

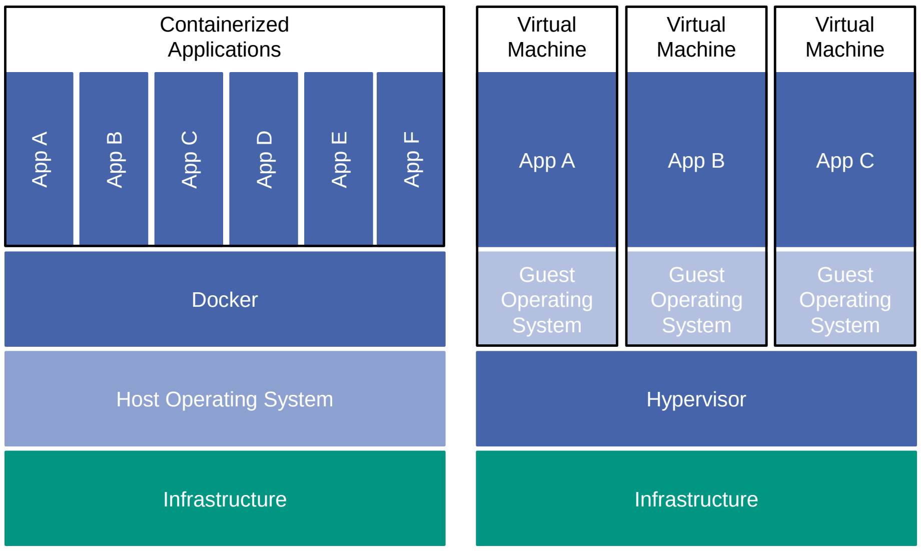


**Figure 2.15.:** Comparing Containers and Virtual Machines taken from [4]

applications from the underlying host operating system. Docker is primarily developed for Linux where it uses built-in resource isolation capabilities.

In order to unlock the full potential of containers, it is necessary to automate the deployment, scaling and management. This process is called container orchestration. In general, orchestration is the automated configuration, coordination and management of computer systems and software [50]. In the context of containers, orchestration means managing the whole lifecycle of containers, including provisioning, deployment, resource allocation, health monitoring, load balancing and configuration. The two main container orchestration systems are Kubernetes 4 and Docker Swarm 5. Since Kubernetes is able to orchestrate not only Docker containers but also other container runtime environments and is used in many production environments, it is a more mature and flexible product. Kubernetes defines several building blocks which are called objects, to provide mechanisms for deploying, maintaining and scaling applications. These objects represent resources that can be easily managed as such. The key objects that are relevant to this work are the following:

- Pod: it is Pod is the smallest building block in the Kubernetes object model. It represents a running process. Inside a Pod, there are one or more running containers. However, a pod has a unique IP address and encapsulates not only containers, but also storage resources and options on how the container(s) should run.
- Service: a Kubernetes service combines a set of Pods that work together. The service acts as an abstraction that redirects requests to the Pods. Therefore, Pods can be dynamically created and destroyed in the background to scale a service.

**Data Exchange and Storage Tools** In this thesis, two different types of tools for data exchange and storage are utilized, one used for the communication between microservices and one for storing persisting data. To realize a very loosely coupled software architecture, an open source in-memory key-value-based database, namely Redis [5] is used as a message broker to realize the publish/subscribe messaging paradigm an store temporary data. An in-memory database is a database management system that, in contrast to classic database management systems, does not rely on the disk storage but rather on the main memory. Therefore, they provide faster and more predictable performance. Indeed, Redis supports only simple and abstract data structures such as strings, hashes, lists, sets, sorted sets, bitmaps, hyperloglogs and geospatial indexes. In other words, complex data structures are not supported leading us to use technologies like JSON (JavaScript Object Notation) to serialize such a structure. For the cluster environment, Redis cluster features high availability and scalability for distributed systems. For the persistence storage, PostgreSQL [6] is used as an object-relational database management system.

[5] redis.io (Accessed: 10.02.2023)
[6] http://postgrest.org/en/v7.0.0/ (Accessed: 10.02.2023)

# 3. State of the Art

Since the introduction of Genetic Algorithms (GAs) in the 60s by Holland [166, 82] and the Evolution Strategies (ES) in the 70s by Rechenberg and Schwefel [154, 169], many frameworks and approaches have been proposed to parallelize and hybridize them. The reasons to add such parallelization and hybridization to EAs are twofold. On the one hand, they alter the numerical behavior of EAs, maintaining a higher genetic diversity for a longer period of time leading to better solutions. On the other hand, they accelerate the computations needed by EAs to find optimal solutions. Recently, parallelization and hybridization of EAs have become even more relevant , since the application of EAs to solve emerging complex and large-scale optimization problems in the industry as well as academic fields has been widely increased. Moreover, modern distributed software technologies such as microservices and container technologies open new perspectives to efficiently parallelize and hybridize EAs on modern hardware such as cluster and cloud environments exploiting their enormous computational power, e.g., [73, 66, 163, 164]. In Section 3.1, an extensive overview of state-of-the-art software architecture and frameworks for parallelizing EAs is introduced. Moreover, different hybridization approaches of EAs and their respective implementations are reviewed in Section 3.2.

## 3.1. Parallelization of EAs

Parallelization of population-based metaheuristics, e.g., EAs have attracted many researchers in the industry as well as in academic domains. Over the years, frameworks and libraries using different software architectures for the execution of parallel EAs have been introduced. These frameworks facilitate the development of new parallel EAs and perform benchmarks quickly. Most of these frameworks follow the object-oriented paradigm for software architecture providing great modularity, reusability and flexibility on the code level.

The four most popular state-of-the-art frameworks for parallel EAs are MALLBA [8, 7], Parallel and Distributed Evolving Objects ParadisEO [33], Distributed Resource Evolutionary Algorithm Machine (DREAM) [145], Distributed Evolutionary Algorithms in Python (DEAP) [62] and JCLEC [181].

MALLBA [8, 7] is a generic library containing algorithms for solving combinatorial optimization problems. The library is written in C++ providing algorithmic skeletons where each one represents a template to realize an EA solving a specific optimization task. Each developed algorithm can be easily executed sequentially and in parallel. Each skeleton provides three different implementations, namely sequential, parallel for Local

Area Networks (LANs) and parallel for Wide Area Networks (WANs). To achieve that, a middleware layer on top of the Message Passing Interface (MPI), called NetStream, for communication in LANs and WANs is provided. NetStream introduces a high-level interface for communication ensuring portability and ease of use of the communication channels. The algorithmic skeletons support the Coarse-Grained Model for parallel EAs. MALLBA introduces a generic skeleton which supports two optimization techniques, namely exact and heuristic. This enables the possibility to hybridize them into one algorithmic approach. For instance, combining Genetic Algorithms (GAs) and Simulated Annealing (SA) can be achieved in two different approaches. While in the first one, the SA algorithm is used to assist the GA by applying the genetic operators, in the second one, the GA is initially run to generate a population which is used by the SA algorithm to evolve the final solution.

ParadisEO [33] is a generic framework dedicated to the reusable design and implementation of parallel metaheuristics. It is a parallel extension of the Evolving Objects (EO) framework [98]. ParadisEO has several similarities with MALLBA regarding the deployed technologies and use cases, i.e., it is written in C++ and uses MPI for communication. ParadisEO has a clear separation between the solution methods and the problem-specific parts. The user only has to adapt the problem-specific parts of the framework while the invariant parts of the solution method remain the same for all problems of an application domain. ParadisEO supports four parallelization models, namely the Coarse-Grained Model, the Global Model, Hierarchical Models and the dimensionality distributed model. Moreover, ParadisEO introduces a hybridization mechanism to combine EA with other methods such as Local Search (LS). In contrast to MALLBA, ParadisEO does not support the execution in WANs.

DREAM [145] is a framework for the automatic distribution of EAs. The framework is designed to be used with WANs where computers are connected through the internet. DREAM uses the Java Evolving Objects (JEO) [13] framework to execute EAs. JEO has similar features compared to the EO library which is deployed in ParadisEO. However, it is written in Java and not in C++. DREAM provides two main levels, namely the high level and the low level. The high level is designed to allow the users to develop EAs, execute and monitor them. It provides a Graphical User Interface (GUI) called GUIDE which allows users with few programming experiences to graphically specify the execution of EA and observe the experiments performed. The EA specification is translated using the easy specification of EAs module from the textual syntax to a Java Code to be executed using the JEO library. At the low level, DREAM has a layer called Distributed Resource Machine (DRM) that provides a distributed P2P mobile agent system that may be used for the automatic distribution of EAs. This architecture allows the execution of parallel EAs without central servers that manage the environment, making it scalable and more effective. DREAM supports only the Coarse-Grained Model for parallelization EAs.

DEAP [62] is an evolutionary computation framework that allows the easy development of EAs. In contrast to the above-mentioned approaches, DEAP allows simple customization of existing algorithms that are developed in DEAP, since it does not hide the implementation details for the user. There is a great focus on usability, since there are high-level abstractions available that facilitate the use of this framework. DEAP is written in the Python programming language making each step of the algorithms, data structures and code easy to read and understand. The core architecture has two modules, namely the creator module and the toolbox module. The creator module is a meta-factory that uses both inheritance and composition to extend existing classes with attributes (data and functions). This mechanism is especially useful for the creation of custom genotypes and populations. The toolbox module is a container for customized tools such as genetic operators. In addition to the previously mentioned modules, DEAP has several peripheral modules like the Distributed Task Manager (DTM) module that allows the distribution of specific parts of the deployed EAs. The DTM is able to distribute certain subtasks across a cluster and to perform load-balancing between the distributed tasks. DEAP is firstly designed to support the Global Model and then it was extended to support the Coarse-Grained Model [149].

JCLEC [181] is a java-based framework which is designed by object-oriented programming principles applying design patterns to maximize the reusability and the ability to hybridize EAs with other algorithms. It provides a high-level software environment to implement any type of evolutionary computing, e.g., GAs and EAs. The framework supports the Global Model by using threads to parallelize the evaluation of each individual. JCLEC is extended to JCLEC-MO to solve multi-objective optimization problems using EAs and PSO [151].

Parallelization of EAs in cluster and cloud environments has recently received a lot of attention of the scientific community. This is due to the fact that many real-world optimization problems are of complex nature due to, e.g., non-linearities, discontinuities, many parameters and restrictions. Moreover, EAs require time-consuming assessments due to the usage of elaborate simulations or the like. Thus, a lot of computational power is needed. The execution of EAs in cluster and cloud environments seems to be a promising step for exploiting their enormous computational capabilities. However, the intrinsic computation and time complexity cannot be reduced by using only cluster and cloud environments but also by the contributing of parallel computing software techniques such as MapReduce, Spark and microservices which allow the solution times to be on reasonable levels.

One of the first implementations to distribute EAs using MapReduce has been introduced by Jin et al. [91]. They proposed the MapReduce for Parallelizing Genetic Algorithms (MRPGA) framework that allows an automatic parallelization of GA. The underlying MapReduce implementation is based on the .NET platform. MRPGA overcomes the problem of mapping the iterative style of EAs to the MapReduce paradigm by adding a coordinator that iteratively starts MapReduce jobs and performs the genetic operators. The mapping phase is dedicated to the evaluation of the individuals while selection takes place in a hierarchical reduction phase that extends the standard MapReduce model. The extended reduction phase has two steps. While in the first step a local selection is applied, in the second one which is called at the end of each generation, the global selection takes place. MRPGA only supports one parallelization model, namely the Global Model. Since the coordinator is a serial component that applies the genetic operators, evaluates the convergence criteria and schedules the whole execution, it rapidly becomes a bottleneck. This limits the scalability of the MRPGA and prevents it from scaling well by using more than 33 nodes. Moreover, the concluded results show that the communication overhead is comparably small, since it is less than 1% of the whole execution time.

Verma et al. [183] improved the MRPGA approach by having only one reduction phase and therefore no change to the underlying paradigm by MapReduce. In contrast to MRPGA, the MapReduce implementation is based on a dedicated framework, namely Hadoop and the local selection takes place in the intermediate phase of the combiner. The mappers still evaluate the individuals and the best is written to the distributed file system to ensure elitism. The default partitioner is overridden, since the default implementation introduces artificial spatial constraints to distribute the key-value pairs to the reducers. Another reason is the premature convergence that skews the distribution of the individuals, since certain individuals will dominate the population at some point. In the reduction phase, all genetic operators are performed and the selection takes place. The client checks the convergence criteria after each generation. The initial population is produced in a separate MapReduce job. The results show that using more computing resources with a fixed size of the optimization problem decreases the time per iteration. There is a decrease in performance if the number of requested nodes exceeds the number of available CPUs.

Salza et al. [165] proposed elephant56, a framework supporting the development and execution of parallel EAs. Elephant56 supports the Global, Fine-Grained and Coarse-Grained Models. It is based on the previous work by Ferrucci et al. [55, 56] that only supports the Coarse-Grained Model. Compared to MRPGA, they share several similarities regarding the Global Model and the Fine-Grained Model. In elephant56 there is also one job per generation and a coordinator that starts the MapReduce jobs, applies genetic operators and performs selection. Therefore, the coordinator in this implementation also becomes the bottleneck. In contrast to the previous work by Ferrucci et al., elephant56 introduces the concept of periods for the Coarse-Grained Model where, in each period, the mapper runs several generations until the migration is performed. This speeds up the computation by reducing the number of MapReduce jobs compared to one job per generation. The migration uses the shuffling mechanism by MapReduce. These two phases of periods and migration are performed in each MapReduce job until the termination criterion has been reached. In their performance assessment [54] of the three deployed parallelization models, they note that the Coarse-Grained Model outperforms the sequential version as well as the Global Model and the Fine-Grained Model for all considered datasets and cluster configurations. This is due to the fact that the overhead of the default data store in Hadoop, namely HDFS, impairs further parallelization improvements gained by the Global Model and Fine-Grained Model.

Martino et al. [45] used the Google App Engine MapReduce framework in order to parallelize GAs in the cloud. Theoretically, the proposed approach supports the Global Model, the Fine-Grained Model and the Coarse-Grained Model. However in their implementation, only the Global Model is tested where the mappers perform the evaluation while the reducers perform the genetic operators. Compared to the other implementations there are two additional components, namely the master and the user program that manage the execution. The master is responsible for the coordination of the resources while the user program performs several tasks like generating the initial population and checking the convergence criteria.

Fazenda et al. [52] proposed a library designed to run EAs in the cloud computing environment based on the MapReduce programming model. The Coarse-Grained Model is chosen to validate the design of such a library. The main drawback of it is that the parallelization introduced by this library is limited to one island. Sherry et al. improved the library by introducing Flex-GP [170] which supports multiple islands. They demonstrated the scalability of up to 350 islands. Llora et al. [119] introduced a framework for semantic-driven data-intensive flows in the cloud. They concluded that Hadoop is a suitable choice to adapt EAs to very large problems as long as the time for each iteration is relatively constant.

Nebro et al. [136] introduced jMetalSP that combines the Big Data analytics engine Spark [189] and metaheuristics implemented with jMetal [48] to parallelize the execution of dynamic multi-objective optimization problems. In jMetalSP, the flexible and extensible architecture of jMetal is combined with the streaming capabilities and the high-level parallel model provided by Spark. While Spark reads information from various data sources, streams it and updates the problem, the jMetal represents the actual solver that solves the optimization problem based on data provided by Spark. The proposed architecture allows jMetalSP to easily run applications on other Big Data platforms, such as Hadoop. The main result of them is that using only one feature from Apache Spark does not introduce a significant enhancement of the performance of jMetal.

Salza et al. [163, 164] proposed a framework, called Advanced Message Queuing Protocol for Genetic Algorithms (AMQPGA), for the execution of the Global Model for genetic algorithms using software containers. The exchange of individuals between the master and its workers is performed by adapting the Global Model to the Advanced Message Queuing Protocol (AMQP) where queues add a level of indirection between the master and its workers. RabbitMQ provides the implementation of AMQP and acts as a message broker to accept and forward messages between the master and the workers. The underlying communication paradigm of AMQP is the publish/subscribe pattern. It allows a potential infinite number of subscribers to listen to one publisher while the publisher does not have to know any of its subscribers. This allows greater scalability compared to direct communication. The master and the workers are executed in Docker containers that manage the model code. CoreOS allocates resources and orchestrates the containers. In their performance test, they varied the execution times of a dummy function between 0.001 ms and 100 ms, the chromosome size between 128 and 65536 and the cluster size between 1 and 128 nodes to measure the communication overhead. They reported an almost linear relationship between the speedup and the cluster size for large evaluation times. The message broker of AMQP becomes a bottleneck if the communication takes a lot of time compared to the evaluation of individuals.

The KafkEO framework proposed by Guervos et al. [73] introduces cloud-based architecture to parallelize EAs. They describe their underlying parallelization model as an asynchronous Coarse-Grained Model. However, the framework is functionally equivalent to the EvoSpace Model [65], since there is no strict topology between the islands. The evolutionary functions are implemented with DEAP [62] and integrated into the serverless framework, namely

OpenWhisk where the deployed services are triggered by arriving messages. Kafka is used as a communication channel that provides message queues. The architecture mainly consists of three services and two message queues. The Producer service creates the initial populations that are sent to a message queue as a new population. The GA Search service gets the population from the message queue, evaluates it and applies the genetic operators to generate offspring. Afterward, the Population Controller service extracts migrants from the evolved population and performs the migration process. This service is the only sequential part of the framework. Therefore, it constitutes a similar bottleneck as the coordinator in the MapReduce implementations.

There are many examples of works that use the Hierarchical Parallel Model of EAs are introduced, e.g., [30, 60, 115, 116, 61]. However, only the works that use modern software or hardware, namely [115, 60] are reviewed in the context of this thesis. Lim et al. [115] presented a framework for Hierarchical Parallel EAs exploiting grid computing. Specifically, the Coarse-Grained Model is combined with the Global Model. For managing the grid architecture, different technologies and tools are used, e.g., The Globus Toolkit, the Commodity Grid Kit, the Ganglia monitoring tool and NetSolve. In the proposed framework, a meta-scheduler is used to perform discovery, bundling and load balancing with the help of gathered information from the clusters by using Ganglia and a Grid Service. The detailed workflow of the proposed framework can be summarized as follows. The meta-scheduler provides the required computing nodes and the grid services for the evolution and evaluation of subpopulations. Then, the subpopulations are transferred to the remote clusters. After that, the parallel evolution is started for the subpopulations. The fitness evaluation of individuals is then performed in parallel over different nodes of the clusters. Once the fitness evaluation has been completed, the obtained fitness is returned to the upper layer where the genetic operations take place. This process is repeated until a specified termination criterion has been reached. Lim et al. reported a speedup gain through the combination between the Coarse-Grained Model and the Global model.

P-CAGE [61] is an environment for the execution of genetic programming, a subclass of EAs which has been implemented as a Peer to Peer (P2P) network. In a P2P network, each participant shares a part of its own resources to provide the service offered by the network [168]. P-CAGE combines the Coarse-Grained Model with the Fine-Grained Model. The islands are distributed on the P2P network nodes and are connected using a bi-directional ring topology. As an epoch termination criterion, a prespecified number of generations has been chosen. For the global termination, three criteria have been implemented, namely the number of epochs, a maximum fixed time and the computation effort. One of them can be chosen before the algorithm starts execution.

García-Valdez et al. [66] implemented evospace-js framework using an event-driven architecture and asynchronous I/O model to parallelize EAs according to the Pool Model. The underlying idea is to use available computing resources in an opportunistic manner. This includes adding and removing resources at runtime. The Evospace Model [65] is the underlying platform of evospace-js. The two main components of the framework are the population repository which is implemented using an in-memory database, namely Redis and the clients called EvoWorkers that execute the actual optimization. García-Valdez et al.

implemented a Particle Swarm Optimization and a Genetic Algorithm using DEAP [62] as EvoWorkers. The EvoWorkers are implemented as microservices communicating with each other via RESTful web services. Meri et al. [131] used synchronous cloud storage services (Dropbox, Sugarsync) to distribute the pool over normal, heterogeneous PCs. Roy et al. [161] used a simple distributed storage to exchange individuals with the pool, while SofEA by Merelo et al. [130] used Apache CouchDB.

Moreover, other approaches for parallel implementations of EAs, e.g. [121, 93, 94, 95] are tied to a specific underlying computing platform such as GPUs. Jurczuk et al. [93, 94, 95] developed and implemented a GPU-based application to distribute evolutionary induction of decision trees for large-scale data applying the Global Model. Luong et al. [121] implemented a Hierarchical Model, namely combining the Coarse-Grained Model with the Global Model on a GPU and CPU platform. The GPU executes the workers of the Global Model, while the CPU executes the Coarse-Grained Model. They identified the communication between the CPU and GPU as a bottleneck.

## 3.2. Hybridization of EAs

Hybridization of EAs with different algorithms and techniques like heuristics and ML techniques paves the road for applying EAs in fields where pure EAs provide inadequate solutions or take too much computational effort to reach an acceptable solution quality. Therefore, the hybridization of metaheuristics – especially EAs – has received a lot of attention by researchers. In this section, we review different approaches to hybridize EAs. These approaches are organized into three categories, namely approaches to assist EAs by generating the initial population, approaches to assist EAs by calculating the fitness evaluation and approaches to assist EAs by applying the genetic operators or by improving the resulting offspring. More in-depth information about most of the research works reviewed in this section can be found in the surveys of Grosan and Abraham m [72], Zhang et al. [190] and Y. Jin [92].

**Population Initialization** Typically, EAs generate the initial population according to some given constraints, randomly. This randomness guarantees a diverse population that assists EAs in exploring the search space. The main advantage of such initialization is that it is simple and does not need any prior knowledge and computationally intensive methods. This facilitates the use of EAs in several domains with minimum adaptation efforts. However, the random initialization of the initial population can mislead the EA by pointing towards unpreferable areas in the search space. Hence, many techniques and algorithms such as heuristics, other population-based metaheuristics, LS and ML techniques are used to assist EAs by generating a more advantageous initial population aiming at decreasing the convergence time and achieving better solution quality [123].

Tseng and Liang [177] proposed a hybrid metaheuristic called ANGEL to solve the Quadratic Assignment Problem (QAP) which is a well-known combinatorial optimization problem. ANGEL combines ACO, GA and LS to solve such problems by finding a permutation of the components that minimizes or maximizes the total assignment cost. Two phases are proposed, namely the ACO phase and the GA phase. Before the GA phase is started, an initial population is generated by the ACO phase. Then, the GA is used to explore the search space and provide promising results to solve the QAP. At the end of the GA phase, the GA returns feedback to the ACO phase via extensive pheromone updating procedures. This pheromone is used by the ACO to evolve a new population for the next run of the GA phase. The LS assists GA and ACO to explore a local neighborhood enhancing the search process of both global algorithms. A similar approach to solve the QAP is used by Vazquez and Whitley [180] by combining GA with Tabu Search. The approach also includes an alternative method for the initial population generation, namely a Greedy Randomized Adaptive Search Procedure (GRASP) [53] which combines greedy elements with random local search elements.

A greedy GA for the QAP is proposed by Ahuja et al. [1]. The proposed algorithm contains several concepts, namely generating the initial population using a randomized construction heuristic, new crossover schemes, an immigration scheme that supports diversity, periodic local optimization of a subset of the population, a tournament among the different populations and a mechanism to achieve a balance between diversity and bias toward better solutions. The greedy GA is applied to all benchmark instances of QAPLIB [31] which is a library of QAP. Out of the 132 instances, the algorithm obtained the best-known solution for 103 instances and found solutions with an enhancement up to 1% of the best-known ones for the remaining 29.

Burke and Smith [32] solved a thermal generator maintenance scheduling problem using a Memetic Algorithm (MA) and compared the obtained results with other algorithms that have a built-in LS as an essential part of their approaches. The used MA uses different LS strategies such as simulated annealing, hill climbing and tabu search. The influence of an initial population on solution quality and computation time is studied. The first results show that the initial population seeded by a heuristic does not introduce a significant influence on the quality of the solution generated by MA. However and in terms of execution time, the seeding of an initial population reduces the running time of MA. Some of the significant benefits are decreasing the time to achieve a given solution quality or discovering better solutions in a fixed time range are observed.

Louis [120] used a case-based reasoning principle to enhance a GA. The intent is to use previous GA runs of similar optimization tasks to seed an adapted initial population for future similar problems. During the evolution process, every individual generated defines a point in the search space and its evaluation provides the respective fitness. The genetic information from the previously computed similar optimization tasks is discarded in basic GAs, but if it is stored in explicit memory, analyzed and then used for the case-based analysis of individuals in population, it can improve the performance of GA. The approach is evaluated for the open shop scheduling problem and yielded consistently better performance than randomly seeded populations.

Zhang and Wang [191] proposed an orthogonal GA with quantization for global numerical optimization with continuous variables. They apply the concept of orthogonal design to two elements during the evolution process, i.e., initial population generation and crossover. Orthogonal design is a kind of statistical method that is used to obtain knowledge from data. Instead of testing all combinations of possible individual constructions or random creations, only the orthogonal combinations are tested by following a statistic-based orthogonal array [110]. Using such an approach enables the generation of an initial population scattered uniformly over the feasible search space so that the GA has a good starting point for the evolution process. The same orthogonal design principles are applied to the crossover operation so that the GA can evenly explore the search space once to locate good points for further exploration in subsequent iterations.

Opposition-Based-Learning (OBL) is a novel ML algorithm inspired by diverse biological, behavioral and natural phenomena [175]. OBL can seed the initial population and further enhance the exploration of the search space. Rahnamayan et al. [148] used OBL in the initialization step and during the evolution process step to enhance a Differential Evolution (DE) algorithm. To apply OBL in the initialization phase, a random initial population is necessary. Afterward, the opposites of the random individuals are calculated and then compared to the randomly generated ones. The fittest of both populations are used as the final initial population for the DE algorithm. The impact of the OBL generated initial population is examined by replacing it with a uniformly generated random population. This approach is evaluated on 15 benchmark problems with 30 and 100 dimensions, including a very large number of local minima. The results indicate that the proposed methods can find the optimal or near-optimal solutions and are more competitive than other comparable algorithms on the studied problems. An acceleration rate up to 57% is achieved.

**Fitness Evaluation** The fitness evaluation is a key step in the evolutionary process of EAs. It assesses the usefulness of each individual to solve a given problem and thereby directly defines the probability that the individual will be selected to generate new offspring for the next generation. For complex real-world optimization problems where the computational cost of the evaluation functions is too high, an estimation of the fitness function is needed. There are numerous approaches for approximating such functions such as ANNs with single-objective function [84, 186] and multi-objective function [76, 124, 75, 15], polynomial regression and Gaussian process [192], radial basis function (previously evaluated nearest-neighbors) [155], heuristics [90, 24] or Markov fitness model [27]. Two exemplary hybridization approaches of using ANN to assist EAs are reviewed.

Wang [186] proposed a hybridization between a GA and ANN to solve the simulation optimization problem of minimizing the total cost of a pressure vessel manufacturing process. Firstly, two approaches are implemented by using only one ANN, i.e., single-ANN and in the second approach with two ANNs, i.e., multiple-ANN. In the single approach, a simple three-layer ANN is designed and trained with 50 feasible samples that are randomly generated based on the considered constraints and an objective function in the limited region. Despite a minor inaccuracy of ca. 10% to 20% between prediction and true performance, the single-ANN approach delivers consistent performances. The multiple-ANN approach uses the same ANN as in the single approach to create multiple ANNs with different random initial weights. The average over the predicted values of each ANN is used to approximate the final fitness value. Thereby, the prediction error is decreased to less than 10% leading to more consistent results.

Magnier and Haghighat [124] proposed an optimization approach based on EA and ANN to solve the multi-objective optimization problem of building design. In this work, numerous decision variables are taken into account to reduce the energy consumption of the building, e.g., window size for passive solar design, heating and cooling temperature set points, relative humidity set points and a supply of air. For simulation purposes, a simulator is developed to estimate the monthly energy consumption of the proposed building design. This model is run using measured weather of the months of January and August 2003 with a two-minute resolution to measure the monthly energy consumption for the heating, cooling and fan of the design. The obtained results of 450 training cases are saved for training the ANN. The trained ANN estimates the monthly energy consumption based on the weather data for each building design proposed by the EA, with an average error up to 1%. The simulator required three weeks to create the training data set. Therefore, if the EA uses the simulator to evaluate each proposed design, it needs ten years to finish the task (based on the number of evaluations).

**Genetic Operators** The performance of an EAs in finding a global optimum for a given objective function depends on the balance between the exploitation of already discovered areas and the exploration of new areas in the search space [173]. These two contradictory requirements are influenced by genetic operators of EAs, namely crossover and mutation. Many different approaches are introduced to hybridize EAs for enhancing, adapting, replacing or creating new genetic operations. For instance, using the Fitness-Blind Mutation for mutation enhancement [173], general purpose or application of problem-specific LS on mutated individuals [77, 86, 167] and using clustering techniques in each generation to select a subset of individuals which are refined by an LS method [125]. In the following, some of the depicted approaches, namely [76, 75, 16] are further explained, since they are relevant to this thesis.

Hacioglu et al.[76, 75] proposed a hybrid EA to solve the inverse airfoil design problem. The essential backbone of this approach is the combination of a GA and ANN. The goal is to design an ANN which maps the highly non-linear relationships between aerodynamic configurations and their corresponding aerodynamic performance to extrapolate or interpolate new candidate solutions from the population. These solutions are added to the population to assist the GA in finding the required design. The ANN predicts solutions based on the gained knowledge during the training process. For this training, the pressure coefficients and distributions serve as input. The output are the design parameters which define the airfoil geometries. The predicted airfoil becomes an individual altering the population pool and participating in the selection process of the parent for the next generation.

Askarzadeh [16] used a more intervening and direct approach to alter the selection of the parents during the reproductive cycle of EAs. Askarzadeh proposed an explicit memory concept to assist a GA, a so-called Memory-based Genetic Algorithm (MGA). It stores the best individuals of the previous runs of GAs to be used later in the next generation. The main goal of the MGA is to prevent premature convergence to suboptimal local optima, especially when the problem is non-linear and there are many local minima in the search space. In the proposed MGA, the best-so-far individual is stored in explicit memory and affects the parent selection process of generating the offspring. This directly impacts all basic genetic operations and consequently enhances the exploitation process of GA. The performance of the suggested algorithm is evaluated on the task of sharing the demanded electrical power among DERs where three wind plants, two photovoltaic plants (PVs) and a combined heat and power system are considered. The main task of MGA is to minimize the billing cost. Simulated results reveal that solutions obtained from the MGA are more accurate than the ones found by the basic GA or PSO algorithms.

For hybridizing EAs with other algorithmic approaches, there are many frameworks and libraries that support such hybridization, e.g., [184, 57, 44, 33, 185, 8, 181] (cf. [126]). Most of them, namely, [184, 57, 44, 185] are designed to allow only the hybridization of EAs without supporting the parallelization. Indeed, only ParadisEO, MALLBA and JCLEC support the parallelization and hybridization of EAs as explained in Section 3.1.

## 3.3. Summary and Discussion

In this chapter, different state-of-the-art research works in the field of parallelization and hybridization of EAs are reviewed. The above-mentioned libraries and frameworks, namely MALLBA [8, 7], ParadisEO [33], DREAM [145], DEAP [62] and JCLEC [181] support the development of new EAs and the parallel execution of them. However, they realize such EAs as monolithic and non-modular applications running on one computer instead of using the computing power of a cluster or a cloud. This is mainly due to the fact that the parallel technologies that are used by these frameworks have several obstacles. For example, Message Passing Interface (MPI) is not designed for generic use in large-scale clusters and cloud environments. Indeed, it handles the underlying computing infrastructure requiring detailed knowledge about it [162]. In contrast to the above-depicted frameworks, frameworks developed based on big data technologies such as MapReduce, Hadoop and Spark [91, 183, 165, 45, 52, 170, 136] provide a highly parallel and scalable runtime environment with full runtime automation in big data environments such as a cluster. Moreover, these technologies represent reliable software environments. Using a high-performance GPU-based platform for executing EAs, e.g., [121, 93, 94, 95] provide far superior performance compared to serial EA implementations. However, GPUs are only able to execute EAs that are specifically adapted to be executed on them. Therefore, they are a lot less flexible than CPU implementations (see, [68]). Another important aspect is the limited scalability of the implementations across several GPUs, since they are mostly limited to one GPU.

The above-mentioned frameworks [8, 7, 33, 145, 62, 181, 91, 183, 165, 45, 52, 170, 136, 121, 93, 94, 95] have a monolithic architecture and therefore do not provide flexibility for interacting with external tools like simulators and a mechanism to integrate an existing EA to a parallel environment. In other words, the EA developers have to implement or adapt their EAs to a specific programming language supported by the framework. For example, EAs must be adapted to the MapReduce programming model for exploiting the full potential of such technologies, forcing EA developers to learn and understand such a programming model. Another obstacle of these frameworks is a lack of modularity at the code level. This complicates the reusability and maintainability of the code during the development process. Moreover, they do not provide an easy-to-use web-based user interface.

Several cloud technologies, such as container virtualization are utilized by [131, 161, 130, 73, 66, 163, 164]. Among them, only [73, 66, 163, 164] use container technologies in combination with a microservice architecture introducing a reliable and scalable approach to parallelize EA. They can integrate any already existing EAs, communicate with external tools and are inherently web-based systems. However, they are designed to support only one parallel model of EA, preventing the application of other parallel models of EAs according to the nature of optimization problems to take their advantages. While the frameworks developed in [163, 164] support only the Global Model, the other two frameworks [73, 66] support the Pool Model. Moreover, they do not provide any flexibility to combine two or more models from the three basic parallelization models of EAs with little adaptation effort for building the Hierarchical Model. The second main drawback of these frameworks is that they do not introduce a simple, generic and efficient mechanism to hybridize EAs with other algorithmic approaches in cluster computing environments. In other words, they do not provide interfaces, i.e., APIs for the integration of any methods to assist existing EAs. Moreover, they do not support the hybridization of parallel EAs in cluster or cloud computing environments.

Alternatively, BeeNestOpt.IAI introduced in this dissertation supports the parallelization of existing EAs according to the basic parallelization models based on three lightweight technologies, namely microservices, container virtualization and the publish/subscribe messaging paradigm without the need to re-implement or re-design them. Precisely, BeeNestOpt.IAI is a software solution that helps EA developers to parallelize their already developed EAs on cluster computing environments with minimal knowledge in distributed systems. BeeNestOpt.IAI is supported with lightweight interfaces (APIs) to achieve such parallelization and to communicate with any external tools or frameworks. Moreover, BeeNestOpt.IAI enables an easy combination of different parallelization models of EAs, a full decoupling between services providing basic building blocks of the algorithm and seamless deployment in a scalable runtime environment such as a cluster. In contrast to most of the aforementioned software frameworks and libraries, BeeNestOpt.IAI – besides the parallelization – enables the hybridization of existing EAs with other algorithms and techniques in a cluster environment. Moreover, BeeNestOpt.IAI supports two ML hybridization approaches, namely the unsupervised hybridization approach and the supervised one.

# 4. Facilitating the Usage of Parallel Evolutionary Algorithms in Cluster Computing Environments

The Scalability of population-based metaheuristics such as EAs represents a main challenge for applying them to complex and large-scale real-world applications [35]. Therefore, parallelization of EAs increasingly becomes popular in tackling this challenge providing satisfying results. The parallel execution of EAs and the division of the population into subpopulations cannot only speedup the computation but also alter the numerical behavior of EAs leading to better solution quality. There are mainly three basic models to parallelize EAs, namely the Global Model (Master-Worker Model), the Coarse-Grained Model (Island Model) and the Fine-Grained Model. Many software approaches are proposed for parallelizing EAs instrumenting different technologies and software techniques as explained in Chapter 3. However, most of these approaches have a monolithic architecture which limits their ability to apply the three parallel basic parallelization models to a given class of optimization problems with minimal adaptation efforts. Moreover, the main disadvantage of these solutions is that the existing EAs cannot be easily plugged into them. Indeed, it is necessary to adapt the existing EAs to their programming paradigm in order to leverage their potential. Other disadvantages are the lack of ability to plug in problem-specific functionality (e.g., simulators) and the restriction to one technology stack.

In this chapter, BeeNestOpt.IAI, a new software solution for simplifying the usage of EAs in high-performance computing environments – such as a cluster – is introduced to answer the research questions **[RQ1]**, **[RQ3]** and **[RQ4]**. BeeNestOpt.IAI allows the parallelizing

Parts of this chapter are reproduced from:

- Khalloof, H., Jakob, W., Liu, J., Braun, E., Shahoud, S., Duepmeier, C., & Hagenmeyer, V. A generic distributed microservices and container based framework for metaheuristic optimization. In Proceedings of the genetic and evolutionary computation conference companion. (2018), pp. 1363–1370. DOI: 10.1145/3205651.3208253
- Khalloof, H., Ostheimer, P., Jakob, W., Shahoud, S., Duepmeier, C., & Hagenmeyer, V. A distributed modular scalable and generic framework for parallelizing population-based metaheuristics. In International Conference on Parallel Processing and Applied Mathematics. (2019), pp. 432–444. DOI: 10.1007/978-3-030-43229-4_37
- Khalloof, H., Ostheimer, P., Jakob, W., Shahoud, S., Duepmeier, C., & Hagenmeyer, V. Superlinear speedup of parallel population-based metaheuristics: A microservices and container virtualization approach. In International Conference on Intelligent Data Engineering and Automated Learning. (2019), pp. 386–393. DOI: 10.1007/978-3-030-33607-3_42

and hybridizing of EAs in a cluster computing environments and provides interfaces for an easy plugin of EAs and third party software components. This solution is developed based on three lightweight technologies, namely microservices, container virtualization and the publish/subscribe messaging paradigm. We start in Section 4.1 by introducing the general parts of the new highly scalable, modular and generic software architecture of BeeNestOpt.IAI for supporting different parallelization approaches. In Section 4.1, the conceptual architecture, the basic services and the general workflow of BeeNestOpt.IAI are presented. In Section 4.2, the Global Model is mapped to the proposed solution for accelerating the execution of EAs where the tasks of the master and workers are carried out by several microservices. In Section 4.3, the architecture of BeeNestOpt.IAI is extended to support the Coarse-Grained Model which reduces the execution runtime, preserves high diversity for a longer time and thus, avoids the problem of premature convergence [143]. The benchmark results for evaluating the performance of BeeNestOpt.IAI considering the Global Model and the Coarse-Grained Model are introduced in Section 4.4. The research contributions presented in this chapter are the main topics of our papers [100, 99, 105].

## 4.1. BeeNestOpt.IAI - Conceptual Architecture for Executing Evolutionary Algorithms in Cluster Computing Environment

The general architecture of BeeNestOpt.IAI shown in Figure 4.1 is designed using a multi-tier structure with the goal of providing abstract interfaces for increasing the flexibility, usability and maintainability of the design as much as possible. It hides the technical aspects of the underlying computing platform by introducing such a layered architecture with two main tiers, namely the User-Interface Tier (UI) in the front-end and the Cluster Tier in the back-end. The UI Tier provides a simple, user-friendly and web-based interface for managing the interaction with the back-end. The UI-Tier supports the user in performing tasks such as defining the input for optimization tasks, uploading optimization models, and starting as well as stopping the optimization tasks. Moreover, it visualizes the obtained results and monitors the whole system by, e.g., showing the usage of CPUs and RAM. The Cluster Tier runs the EAs on a cluster to solve optimization tasks and stores the required data. It contains two sub-layers, namely the Container Layer and the Data & Message Layer. The Container Layer is designed based on microservices and container technologies providing high flexibility, modularity and generality in executing already developed EAs on a cluster. Compared to the monolithic applications, a microservice-based architecture enforces some useful properties to facilitate the usage of parallel EAs in a cluster computing environment. In other words, an application featuring the microservices architecture is composed of several independent services, where each service performs a special task which it is exposed via a lightweight communication interface [139]. Hereby, each service can use its own technology stack for the implementation. A valuable advantage of microservice lies in its

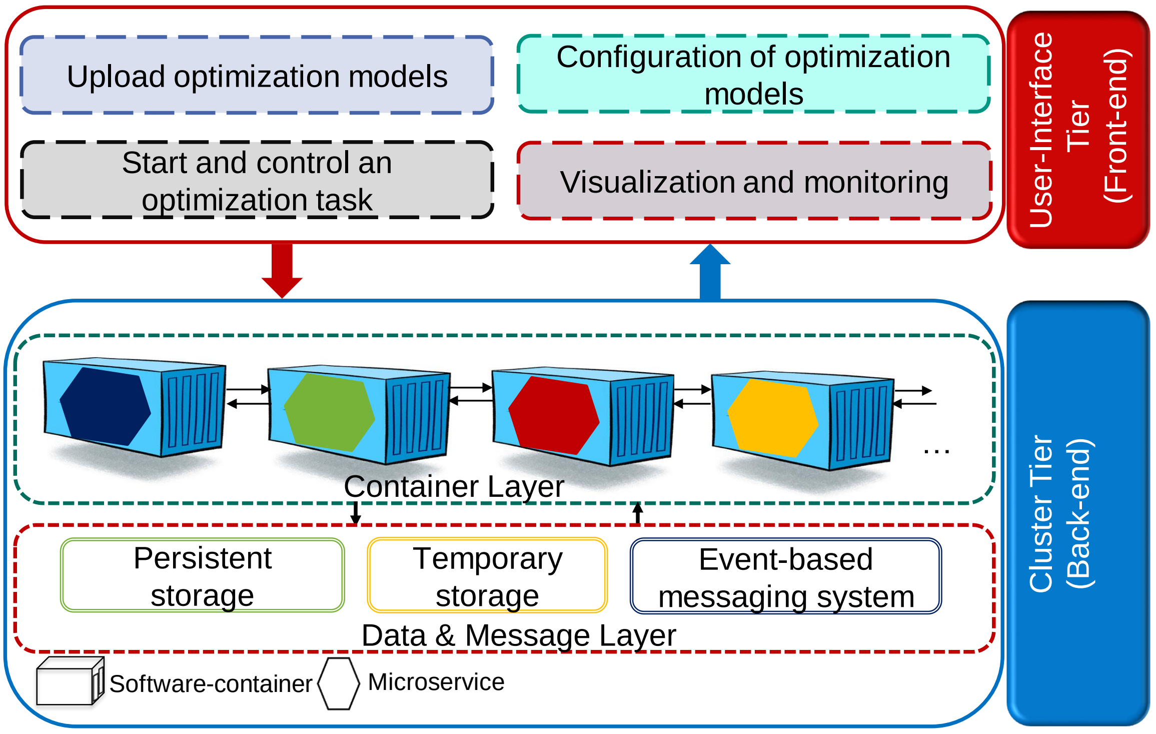

Upload optimization models
Configuration of optimization models
Start and control an optimization task
Visualization and monitoring
User-Interface Tier (Front-end)
...
Container Layer
Persistent storage
Temporary storage
Event-based messaging system
Data & Message Layer
Software-container
Microservice
Cluster Tier (Back-end)

**Figure 4.1.:** The conceptual architecture of BeeNestOpt.IAI

ability to build highly parallel and scalable solution where each service can use the most suitable technologies and be scaled horizontally. Moreover, each service is encapsulated at runtime in containers which are dynamically created on demand for runtime automation and platform independence. Combining microservices with container runtime automation unlocks the full potential of EAs by allowing the execution of them on large-scale computing clusters. The architecture of the Container Layer supports EA developers by providing an implementation of different parallelization models of EAs. It allow plugging in their own EA implementation as a separate service, hybridizing EAs with other algorithms to form HEAs and incorporating EAs into a more complex system. In such systems, different tools and simulators can cooperate together to solve an optimization task. The Container Layer contains all services necessary to execute EAs on a cluster. This not only includes services that actually execute the EA, but also services that coordinate the execution and distribute the data. In other words, the main functions provided by BeeNestOpt.IAI, namely parallelizing EAs, hybridizing EAs to support HEAs, coordinating the execution, distributing the necessary data, managing containers and starting external simulators are carried out in the Container Layer. The Data & Message Layer stores data required by EAs for solving an optimization problem and also the obtained results of each optimization job. Moreover, it serves as an intermediate storage for message exchange among services. To reflect these functionalities, the Data & Message Layer is subdivided into a persistent storage, a temporary storage and an Event-based messaging system. The persistent storage is responsible for storing the final results and the data required for performing one optimization task. The temporary storage stores the intermediate data that is exchanged between the services during the execution of an optimization job. The Event-based messaging system realizes a publish/subscribe message communication pattern to improve the decoupling among the services by enforcing event based communication between the services. For facilitating the communication among the services within the Container Layer and between the layers, two communication styles, namely the request/response and the event-based (i.e., publish/subscribe) are used. On the one hand, REST (REpresentational State Transfer) APIs are a common lightweight implementation of the request/response communication style. It is used for the synchronous communications that occur at the beginning and end of each optimization job. On the other hand, the publish/subscribe messaging paradigm is an implementation of the event-based communication style. It is used for exchanging message among the services during the run-phase of an optimization job. Utilizing microservices, container technologies and the publish/subscribe messaging paradigm within the Container Layer ensures a seamless deployment, full decoupling among the services and increase the flexibility of the design to provide several parallelization models of EAs by BeeNestOpt.IAI in one single framework. In the following sections, the term service refers to microservice.

BeeNestOpt.IAI distinguishes between two groups of services namely, the basic services and the model-related services. While the first group realizes the general functionalities required by the parallelization and hybridization models of EAs, e.g., the ones related to communication and computing infrastructure, the second group implements the functionalities related to each parallelization and hybridization model. The basic services are defined abstractly, so that each one can be modified and reused according to the considered parallelization or hybridization model. By realizing any parallelization or hybridization model of EAs into BeeNestOpt.IAI, we add the model-related services that specially perform the tasks of this model into the Container Layer. For the basic services, we either modify their functionalities or add new ones. This increases the flexibility, generality and modularity of BeeNestOpt.IAI and facilitates the realization of different parallelization and hybridization models of EAs in a cluster computing environment. The general execution workflow of EAs in BeeNestOpt.IAI has three stages, namely the ‘Initialization Phase’, the ‘Iterative Evolution Phase’ and the ‘Termination Phase’. In the following section, we describe the basic services of BeeNestOpt.IAI and the general execution workflow.

### 4.1.1. Basic Services of BeeNestOpt.IAI

Executing EAs in a highly distributed environment such as a cluster using microservices and container technologies implies several general tasks such as creating the required services to carry out an optimization job in parallel and fetching required data from a database. To this end and as shown in Figure 4.2, three decoupled and cohesive microservices within the Container Layer of BeeNestOpt.IAI are designed.

- The Optimization Job Management Service (Opt.J.M. Service).
- The Container Management Service (Co.Ma. Service).
- The Data & Message Service (Da.Me. Service).

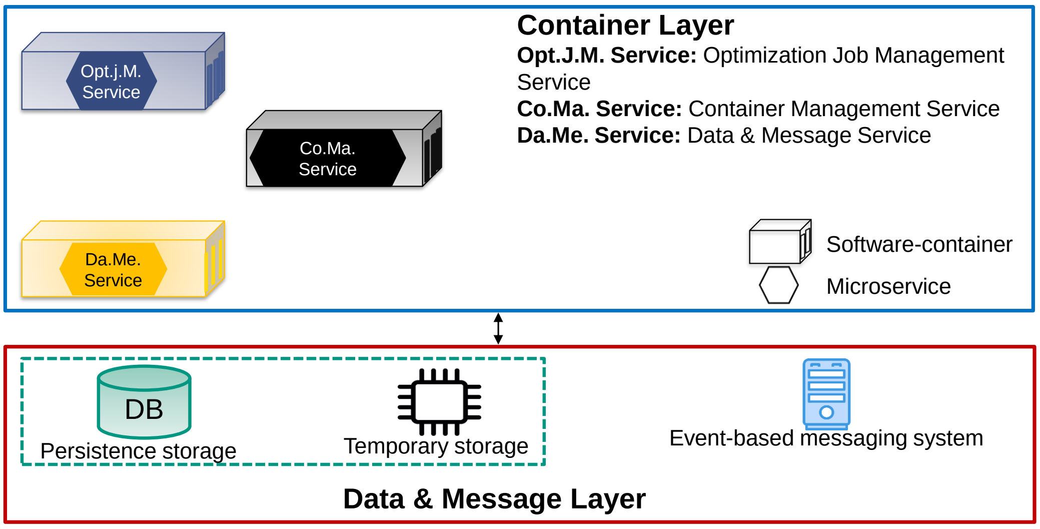

Container Layer
Opt.J.M. Service: Optimization Job Management Service
Co.Ma. Service: Container Management Service
Da.Me. Service: Data & Message Service
Opt.j.M. Service
Co.Ma. Service
Da.Me. Service
Software-container
Microservice
DB
Persistence storage
Temporary storage
Event-based messaging system
Data & Message Layer

**Figure 4.2.:** The basic microservices within the Container Layer of BeeNestOpt.IAI

**Optimization Job Management Service (Opt.J.M. Service)** The Optimization Job Management Service (Opt.J.M. Service) is the central (user) access point for creating, tracking, monitoring and managing optimization jobs as well as coordinating the interaction between the services in the back-end Tier and the user. The Opt.J.M. Service receives the configurations for each job from the UI Tier as a JSON structure. It interprets them and extracts three main configurations, namely the cluster configuration, the optimization job configuration and the EA configuration. The cluster configuration contains information about, e.g., the number of CPUs and size of RAM required for running the microservices and their containers in a cluster computing environment as shown in Figure 4.2. The optimization job configuration defines all configurations related to each optimization job like the parallelization model, objective functions, required data and type of the communication among services, i.e., synchronous or asynchronous. The EA configuration holds all EA configurations related to the actual optimization job such as population size, acceptance policy and termination criteria, to name a few. The Opt.J.M. Service extracts these three type of configurations and sends them to the corresponding services to be applied.

The Opt.J.M. Service exposes a REST API to run several optimization jobs with different configurations as batch jobs (i.e., workloads) on the microservice runtime environment (i.e., a Kubernetes cluster). It waits until all other services signalize that they have applied their specific configuration parameters and are ready to perform their job for an optimization job. The Opt.J.M. Service receives information from the underlying cluster computing environment about the usage of CPUs and RAM. The status of each optimization job, namely ‘Initialized’, ‘Running’, ‘Stopped’ or ‘Finished’ is handled by the Opt.J.M Service. The current and final results of each optimization job are also sent to the Opt.J.M. Service which in turn sends them to other services to be stored. Table 4.1 lists the REST-APIs offered by the Opt.J.M. Service.

In addition to the functionalities presented in Table 4.1, the Opt.J.M. Service also creates a group of publish/subscribe channels for each optimization job in the Event-based messaging

| Method | URL-Pattern | Description |
|---|---|---|
| POST | /opt/jobs | A POST request on this URL is used to submit the configurations for several optimization jobs. When the required services are started and ready to be used, the Opt.J.M. Service returns a list of optimization job descriptions containing a unique job-id and the status 'Initialized' to the client. In case of an error, an error description will be returned as a JSON structure. |
| POST | /opt/job/{job-id}/ start/ opt/job/{job-id}/stop | The API methods allow to start or stop an optimization job. After the first step of starting the required services for an optimization job, the start API can be called to start an optimization job. A job status JSON structure will be returned which describes if the status change could be achieved. The service returns the status 'Running' if an optimization job is successfully started. The status 'Stopped' is returned as a response if an optimization job is stopped without any trouble. In case of an error, an error description will be returned as a JSON structure. |
| GET | /opt/job/{job-id} | A GET request to this URL is used to retrieve all information related to an optimization job, e.g., the status 'Initialized', 'Running', 'Stopped', or 'Finished'. The latter one is returned when an optimization job stops itself after reaching one of the predefined stopping criteria. Besides status information, the applied cluster configuration and EAs configuration related to an optimization job are also returned. |
| GET | /opt/job/{job-id}/ results | A GET request to this URL is used to obtain current results, i.e, either intermediate or final results of an optimization job. If the optimization job is running, intermediate results are returned. If an optimization job has one of the statuses 'Stopped' or 'Finished', the final results are returned. |
| DELETE | /opt/job/{job-id} | Destroy the runtime environment of an optimization job with the given id and free any resources it consumes. |

**Table 4.1.:** The APIs of the Opt.J.M. Service

| Method | Address | Description |
|---|---|---|
| SUBSCRIBE | status.service.{serviceName}.job.{job-id} | Get the status of each service, i.e., 'Initialized', 'Running', 'Stopped' or 'Finished' for an optimization job. |
| SUBSCRIBE | result.job.{job-id}.type.{resultType} | Get intermediate or final results (referred to as {resultType}) of an optimization job. |
| PUBLISH | start.job.{job-id} | Publish a start signal to start an optimization job. |
| PUBLISH | stop.job.{job-id} | Publish a stop signal to stop an optimization job. |

**Table 4.2.:** The publish/subscribe channels of the Opt.J.M. Service

system or subscribes to some channels as shown in Table 4.2. This channel infrastructure is used for event-based coordination between different service instances that are interacting while performing tasks of an optimization job. The microservices of BeeNestOpt.IAI post information about their status changes and other possible information on their own channels when performing tasks for the given optimization job. Additional information is also sent to the Opt.J.M. Service. Such information includes the availability of certain intermediate results or final ones of an optimization task and the status information about the evolution of the populations by the deployed EA. This enables the Opt.J.M. Service to monitor the whole optimization process, e.g., to know in which status the executed optimization is existing.

The design flexibility of the Opt.J.M. Service enables the possibility to implement new APIs and publish/subscribe channels for achieving a seamless interaction between EAs and external tools or frameworks like forecasting frameworks as shown in Figure 4.3.

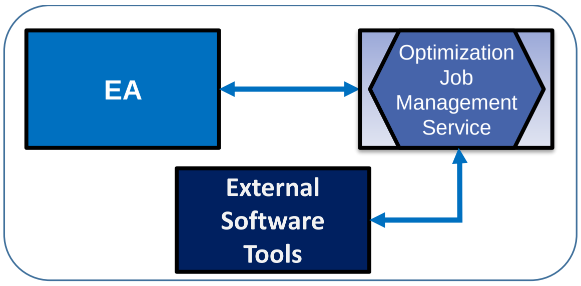


**Figure 4.3.:** Supporting the interaction between EAs and external tools

**Container Management Service (Co.Ma. Service)** The Container Management Service (Co.Ma. Service) is used to automatically create the required containers running services for an optimization job based on the cluster configuration and the optimization job configuration extracted by the Opt.J.M. Service. Before deploying new containers for any service, the Co.Ma. Service checks whether there are already deployed instances that are not used by another optimization job. If this is the case, the Co.Ma. Service resets and

| Method | URL-Pattern | Description |
|---|---|---|
| POST | /com/ job/ {job-id}/create | A POST request on this URL is used to create the required containers according to the description in the payload which is extracted from optimization job and cluster configurations. A list of information about status and access to those containers after creation is returned. |

**Table 4.3.:** The APIs of the Co.Ma. Service

configures them for a new optimization task. Otherwise, it creates them from scratch. Table 4.3 describes the APIs the Co.Ma. Service.

**Data & Message Service (Da.Me. Service)** The Data & Message Service (Da.Me. Service) is the central access point to the persistence and temporary storages of the Data & Message Layer. It stores required data for performing an optimization job and all information related to an optimization job, e.g., the status of the cluster computing environment and services, final results and configurations in the persistence storage. Temporary data exchanged among services, e.g., subpopulations and intermediate results are stored in the temporary storage. The Da.Me. Service is responsible for retrieving such data to be used by other services or tiers such as UI Tier. Table 4.4 describes the main APIs of the Da.Me. Service.

In an underlying cluster computing environment (e.g., Kubernetes), the aforementioned three basic services namely, the Opt.J.M. Service, the Co.Ma. Service and the Da.Me. Service as well as the components of the Data & Message Layer will run permanently as daemon processes. It is noteworthy that the three basic services and their APIs are generally designed, so that their functionalities can be modified or new ones can be added according to the considered parallelization model as we will see in the following sections and chapters.

Continuously, the above basic and all model-related services – explained later – publish their status on the status-channel of the publish/subscribe channel infrastructure as shown in Table 4.5.

## 4.1.2. General Execution Workflow

The general execution workflow starts with the 'Initialization Phase' which is carried out once per optimization task, then the 'Iterative Evolution Phase' is iteratively performed and finally the 'Termination Phase' takes a place. In the following sections, we describe how the basic microservices are involved in each phase of the execution workflow of EAs in BeeNestOpt.IAI.

| Method | URL-Pattern | Description |
|---|---|---|
| POST | /dam/job/{job-id}/ data/{data-type}/store | A POST request on this URL is used to store both considered data types (referred to as {data-type}), i.e., persistence and temporary data. For example, data required for solving an optimization problem and other data related to an optimization job, namely final results and configurations stored in the persistence storage. Intermediate results and subpopulations are stored in the temporary storage. This API is abstractly designed to handle a wide range of data, e.g., time series data or structured data. |
| GET | /dam/job/{job-id} | A GET request to this URL is used to retrieve all information related to an optimization job from the persistence storage such as status, configurations and results. |
| GET | /dam/job/{job-id}/ data/{data-type} | A GET request to this URL is used to retrieve a specific type of data required for solving an optimization process. This API is also designed to be as general as possible. |

**Table 4.4.:** The APIs of the Da.Me. Service

| Method | Address | Description |
|---|---|---|
| Publish | status.service.{serviceName}. job.{job-id} | Publish the status of the services, i.e., 'Initialized', 'Running', 'Stopped' or 'Finished' for an optimization job. |

**Table 4.5.:** The status-channel of BeeNestOpt.IAI

**Initialization Phase** It starts by sending the configurations as a JSON file from the UI Tier to the Opt.J.M. Service. Firstly, the Opt.J.M. Service processes the input JSON which contains configurations for several optimization jobs and then assigns a job-id for each optimization job. This identifier is included in each interaction among services to identify, e.g., cluster configuration, EA configuration, optimization job configuration, results and the location of required data related to each optimization job. After that, the Opt.J.M. Service sends the cluster configuration and a part of the optimization job configuration, e.g., the parallelization model to the Co.Ma. Service to create a new optimization job environment for each job according to these configurations. Indeed, it starts the necessary runtime artifacts for this job, namely the model-related services which publish the 'Initialized' status to the status-channel when they are started. At the end of this phase, the Opt.J.M. Service sends the applied configurations for each optimization job with its job-id to the Da.Me. Service to store them. The UI Tier can every time access this data and visualize it.

**Iterative Evolution Phase** After initializing the required services, a start signal – for at least one optimization job – is sent from the UI Tier to the Opt.J.M. Service which in turn asks other services to start the required processes to perform the selected optimization job. At the end of each iteration, the Opt.J.M. Service fetches the aggregated results for monitoring purposes.

**Termination Phase** Once one of the termination criteria is fulfilled, this phase is carried out. A stop signal is sent to the Opt.J.M. Service for stopping the whole process. This stop signal can be sent either from model-related services or from the UI Tier. In both cases, the Opt.J.M. Service sends the current obtained results and its related population to the Da.Me. Service to store them into the persistent storage for later access, e.g., for visualization by the UI Tier.

## 4.2. Mapping the Global Model to BeeNestOpt.IAI Architecture

In this section, the extension and the adaptation of BeeNestOpt.IAI architecture for parallelizing EA according to the Global Model in a cluster computing environment is presented. The model-related services and the adapted basic ones are described in Sections 4.2.1 and 4.2.2, respectively. To understand the mapping of the Global Model to the BeeNestOpt.IAI architecture, the execution workflow required to achieve the main functionalities of the services is presented in Section 4.2.3.

### 4.2.1. Global Model-related Services

For mapping the Global Model to the proposed software architecture, the aforementioned three basic services of BeeNestOpt.IAI are partly adapted to support the Global Model. Moreover, the following three new model-related services are designed to interact with the basic ones for performing the task of the Global Model as shown in Figure 4.4.

- Evolutionary Operators Service (E.O. Service).
- Distribution & Synchronization Service (Ds.S. Service).
- Calculation Service (Ca. Service).

In contrast to the basic services, the E.O. Service, Ds.S. Service and Ca. Service instances are created dynamically on the underlying microservice runtime environment for each optimization job at job creation time. The type of EA algorithm, the corresponding E.O. Service and the worker processes that are used are depending on the actual Master-Worker optimization job configuration.

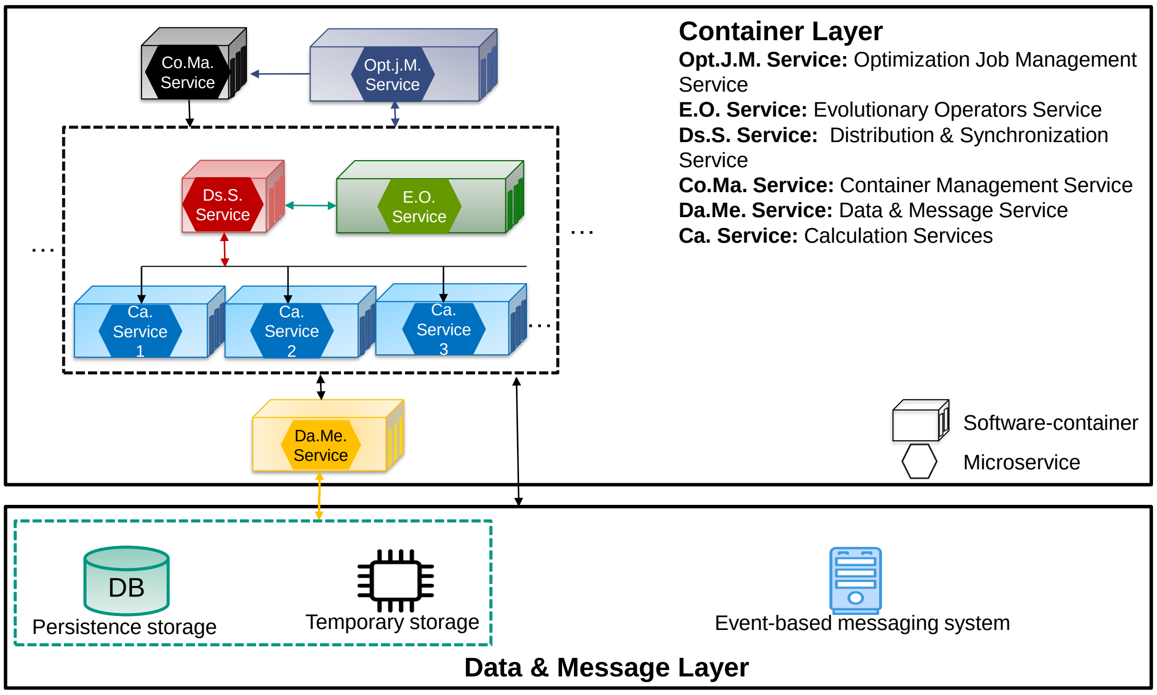
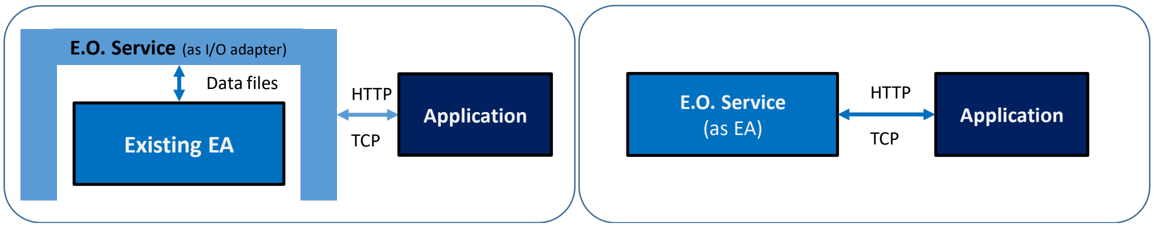

Co.Ma. Service
Opt.j.M. Service
Ds.S. Service
E.O. Service
Ca. Service 1
Ca. Service 2
Ca. Service 3
Da.Me. Service
Container Layer
Opt.J.M. Service: Optimization Job Management Service
E.O. Service: Evolutionary Operators Service
Ds.S. Service: Distribution & Synchronization Service
Co.Ma. Service: Container Management Service
Da.Me. Service: Data & Message Service
Ca. Service: Calculation Services
Software-container
Microservice
DB
Persistence storage
Temporary storage
Event-based messaging system
Data & Message Layer

**Figure 4.4.:** Mapping the Global Model to the cluster tier of BeeNestOpt.IAI

**Evolutionary Operators Service (E.O. Service)** In the Global Model, the genetic operators, namely crossover, mutation and selection operation are performed by the master as described in Section 2.2.1. The E.O. Service acts as a master by generating the population, applying the genetic operators and selecting the parents of the offspring as well as the surviving offspring for the next generation. Such tasks can be implemented in any programming language according to the EA implementation concepts of a certain EA developer which can be quite different. An E.O. Service integrable into BeeNestOpt.IAI can be designed as a distinguished microservice as shown in Figure 4.5b which implements the APIs listed in Table 4.6 and subscribes to channels described in Table 4.7. Another possibility is to use the already implemented microservice which only implements the E.O. Service APIs shown in Table 4.6 and subscribes to channels described in Table 4.7. It defines an Input/Output (I/O) adapter interface (see, Figure 4.5a) that provides the ability to encapsulate any already developed EA as it exists via, e.g., its command and file system interfaces (if there are any) to perform the aforementioned tasks. Precisely, an external EA application is used as it exists and should not be adapted, since the pre-given E.O. Service uses glue code which interfaces with the command and file system interfaces of the external EA application. This enables the integration of already developed EA into the microservice-based platform, namely BeeNestOpt.IAI without any changes. The I/O adapter (e.g., glue code) realizes a bridge between the deployed EA and other services providing two main functions. Firstly, the transformation of external requests into understandable ones to be applied by the deployed EA. Secondly, the reaction of the deployed EA on these requests into acceptable form by external applications. Precisely, it receives the requests including EA configuration from external applications using its API and translates them in a way that the used EA can understand them and react to them. Besides requests, it iteratively



**(a)** The generic E.O. Service as adapter **(b)** EA as a standalone microservice implementing the E.O. Service APIs

**Figure 4.5.:** The concept of the E.O. Service to integrate already developed EA into BeeNestOpt.IAI

receives the results of the evaluation process from the Opt.J.M. Service and formats them for matching the original list format supported by the EA. These results are required for calculating the fitness function of each individual and consequently applying the genetic operators.

| Method | URL-Pattern | Description |
|---|---|---|
| PUT | /eo/job/{job-id}/ configuration | A PUT request on this URL is used to set up the EA configuration of the E.O. Service instance before the deployed EA is actually started. This can include, e.g., the chromosome model and runtime parameters for the EA. |
| GET | /eo/job/{job-id}/ status | A GET request on this URL sended from the Opt.J.M. Service is used to receive the current status of the EA process. The status information can contain information like the status of the EA: ‘Running’, ‘Stopped’, ‘Finished’. It also contains the count of iterations performed or a summary of statistical data about the intermediate status of the evolution. |
| POST | /eo/job/{job-id}/ result | A POST request on this URL is used to submit the intermediate and final results of the evaluation process of individuals. |

**Table 4.6.:** The APIs of the E.O. Service for the Global Model

After applying the configurations included into the requests and evaluating the intermediate results by the used EA, the I/O adapter reads the output of EA, i.e., the population or offspring and transfers them back in standardized formats to be used by the other services as shown in Figure 4.5.

The E.O. Service acting as the master of the Global Model starts the evolution process of this specific optimization job after receiving a start signal from the Opt.J.M. Service using the publish/subscribe channels presented in Table 4.7.

| Method | Address | Description |
|---|---|---|
| SUBSCRIBE | job.{job-id}.start | Get a start signal to start an optimization job by generating a population according to the applied EA configuration. |
| SUBSCRIBE | job.{job-id}.stop | Get a stop signal to stop an optimization job. |

**Table 4.7.:** The publish/subscribe channels of the E.O. Service for the Global Model

**Distribution & Synchronization Service (Ds.S. Service)** The Ds.S. Service is a central component for parallelizing the Global Model. It splits the population sent by the E.O. Service evenly into subpopulations that are distributed over the workers, namely the Ca. Service instances, for evaluation. After receiving a start signal from the Opt.J.M. Service and depending on the defined Master-Worker synchronization policy (see, Section 2.2.1), the Ds.S. Service sends the subpopulations either to all workers once, in case of synchronous mode or to each one separately, in case of asynchronous mode. In the first mode, the Ds.S. Service collects the partial results and waits until all workers finish their jobs to join the results. However, for the last one, it collects and joins partial results as soon as they are available, i.e., it does not wait until all workers are finished from their calculations. Finally, it sends the built result list to the Opt.J.M. Service for further usage. The main functionality of the Ds.S. Service API is described by the URL patterns in Table 4.8.

| Method | URL-Pattern | Description |
|---|---|---|
| POST | /ds/job/{job-id}/configurations | A POST request on this URL is used to submit the optimization job configuration, e.g., worker configuration and synchronization mode related to the current optimization task to be distributed over corresponding services. |
| POST | /ds/job/{job-id}/population | A POST request on this URL is used to submit a list of chromosomes forming a population to be split. |

**Table 4.8.:** The APIs of the Ds.S. Service for the Global Model

The Ds.S. Service stores the subpopulations in the Event-based messaging system, i.e., the publish/subscribe channel infrastructure provided by the Data & Message Layer to be read subsequently by the Ca. Service instances. The workers also use the same publish/subscribe channel infrastructure for publishing the calculation results. The Ds.S. Service subscribes to the result-channels to build the intermediate results when an optimization job is running and the final ones when all workers finish their calculations. The Ds.S. Service has the publish/subscribe channels described in Table 4.9.

| Method | Address | Description |
|---|---|---|
| PUBLISH | population.job. {job-id}.worker. {worker-id} | Publish the subpopulations to each worker for an optimization job. |
| PUBLISH | config.job.{job-id}. worker.{worker-id} | Publish configurations to each worker for an optimization job. |
| PUBLISH | result.job.{job-id}. type.{resultType} | Publish the joined intermediate or final results for an optimization job. |
| SUBSCRIBE | result.part.job.{job-id}. worker.{worker-id} | Get the partial result related to each worker. |
| SUBSCRIBE | stop.job.{job-id} | Get a stop signal to stop an optimization job. |

**Table 4.9.:** The publish/subscribe channels of the Ds.S. Service for the Global Model

**Calculation Services (Ca. Service)** The Ca. Service instances act as workers in the Global Model that get the individuals in form of subpopulations and calculates the values of the objective functions and constraints defined by the user for each individual. In some cases, this calculation requires some data which is provided by the Da.Me. Service. When a job is started, as many instances from the Ca. Service are created as required by its configuration. Each Ca. Service reads the subpopulation assigned to it from its corresponding publish/subscribe channel. It evaluates it and publishes the results in its own publish/subscribe channel created for the considered job. The Ds.S. Service gets the results of each worker and then joins them with other partial results coming from the other workers. The publish/subscribe channels used by each Ca. Service are listed in Table 4.10.

| Method | Address | Description |
|---|---|---|
| SUBSCRIBE | population.job. {job-id}.worker.{worker-id} | Get the input, i.e., the subpopulations to be evaluated by calculating the objective functions and constraints. |
| SUBSCRIBE | config.job. {job-id}.worker.{worker-id} | Get the configurations, e.g., the objective function and constraints related to an optimization job |
| PUBLISH | result.part.job. {job-id}.worker.{worker-id} | Publish the partial result of the evaluation. |
| SUBSCRIBE | stop.job.{job-id} | Get a stop signal to stop an optimization job. |

**Table 4.10.:** The publish/subscribe channels of the Ca. Service for the Global Model

Any software tool such as a simulator can act as a Ca. Service by implementing the publish/subscribe channels showed in Table 4.10.

### 4.2.2. Adaptation of the Basic Services to Support the Global Model

Some of the basic functionalities of the Opt.J.M. Service and Co.Ma. Service described in Section 4.1.1 are adapted to support the Global Model. The functionality of the Opt.J.M. Service for sending the extracted configurations to the model-related services is defined as follows.

Firstly, the Opt.J.M. Service sends the cluster configuration to the Co.Ma. Service which is adapted to create the necessary containers of microservice instances of the Global Model according to these configurations. Those are an instance of the Ds.S. Service, the E.O. Service running a certain EA as a master and on demand the required containers for the Ca. Service (Workers) in the required quantity. The Ca. Service instances – realized as container images implementing a Ca. Service REST interface – are needed for evaluating the objective functions and the constraints of the optimization problem. The Co.Ma. Service assigns an identifier (worker-id) to each worker by the creation. This identifier is included in the name of each publish/subscribe channel of the worker to facilitate the communications between it and other services. After that, the Opt.J.M. Service sends the EA configuration to the E.O. Service to be applied and the configurations extracted from the optimization job configuration which is related to the Ds.S. Service and workers to the Ds.S. Service. The Ds.S. Service applies its own configuration, e.g., the type of communication between the master and workers, i.e., synchronous or asynchronous and distributes the configurations related to workers over them. These configurations include, e.g., the simulation model, the objective functions, the constraints and the location of data in the persistence storage required to solve the considered optimization problem. The Da.Me. Service using the components of the Data & Message Layer supports the proposed microservices to execute the Global Model. The Da.Me. Service with its APIs explained in Table 4.4 is used to store the final found solutions (individuals) and all information, e.g., configurations and results for an optimization job into the Data & Message Layer as explained above in Section 4.1.1.

### 4.2.3. Execution Workflow

To execute an optimization task using an EA parallelized according to the Global Model in the cluster environment, the microservices and the Data & Message Layer cooperate together. They follow the three general execution workflow phases explained in Section 4.1.2. In the following sections, we explain the three phases of the execution workflow for the Global Model and the role of each model-related service in each phase.

**Initialization Phase** The basic service of BeeNestOpt.IAI performs the general 'Initialization Phase' explained in Section 4.1.2. After booting up the required services like the E.O. Service, the Ds.S. Service and the instances of the Ca. Service, the Opt.J.M. Service starts sending the configurations of the optimization job to the Ds.S. Service to be distributed over the other services like Ca. Service instances. Moreover, it sends also the EA configuration to the E.O. Service to be applied as described above. After finishing the 'Initialization Phase', the E.O. Service, the Ds.S. Service and the instances of the Ca. Service publish the 'Initialized' status to the status-channel.

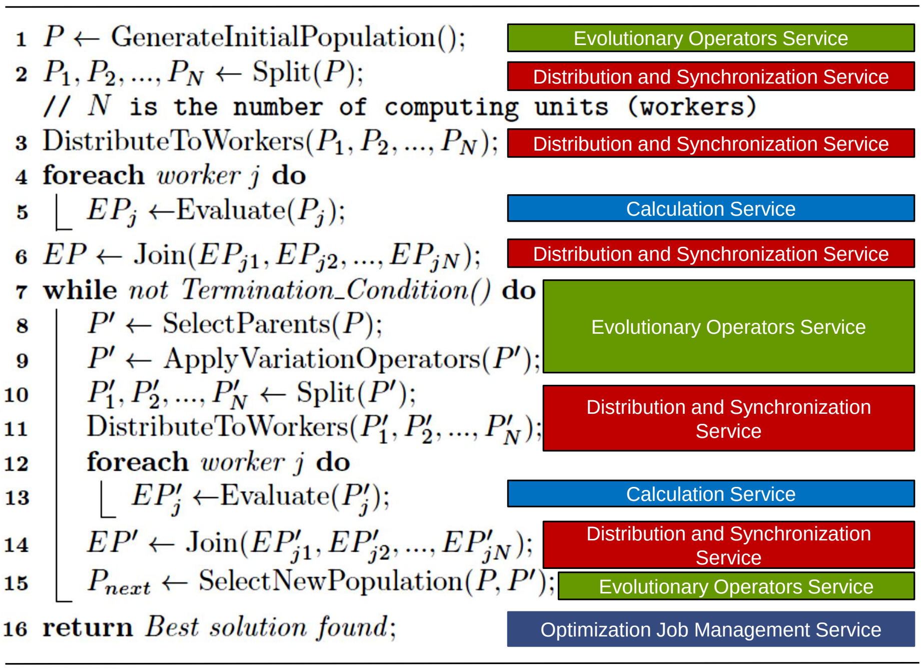
**Iterative Evolution Phase** A start signal is sent from the UI Tier to the Opt.J.M. Service which in turn forwards it to the E.O. Service (master). The E.O. Service receives this signal and starts generating the chromosomes as a population according to the applied EA configuration and sends them to the Ds.S. Service. Each population is split up by the Ds.S. Service into smaller sized parts (the size will be as equal as possible) according to the number of available workers, i.e., the Ca. Service instances and the parts are then published into the corresponding channel of each worker. Then, the Ca. Services are informed via the Event-based message system that new data for evaluation is available. Each Ca. Service fetches the assigned subpopulation from its channel, calculates the values of the assessment criteria for each individual and writes the results to the result channels of the Event-based message system notifying the Ds.S. Service that a result is available. In a synchronous Global Model, once all Ca. Service instances have finished, the partial results are read and merged together in one list by the Ds.S. Service. By applying an asynchronous Global Model, the Ds.S. Service builds the partial results when at least one Ca. Service instance is finished. The Ds.S. Service returns the merged result to the E.O. Service to evaluate the quality of generated individuals and apply the generic operator. If no termination criterion such as the quality of solution, the elapsed time or the performed generations is met, the optimization process continues. The E.O. Service performs the genetic operators, namely selection, crossover and mutation to generate the offspring to be processed over and over until a termination criterion is reached. Furthermore, the merged result is sent to the Opt.J.M. Service which in turn sends it to the Da.Me. Service to be stored.

**Termination Phase** A stop signal either from the E.O. Service or from the UI Tier is sent to the Opt.J.M. Service for stopping the master and workers as explained in Section 4.1.2. Table 4.11 summarizes how the above microservices are involved in the three stages of the execution workflow.

| Services | Initialization Phase | Iterative Evolution Phase | Termination Phase |
|---|---|---|---|
| Opt.J.M. Service | Yes | No | Yes |
| E.O. Service | Yes | Yes | Yes |
| Ds.S. Service | Yes | Yes | No |
| Co.Ma. Service | Yes | No | No |
| Ca. Service | Yes | Yes | No |

**Table 4.11.:** Mapping the microservices of the Global Model to the three execution phases

The above three model-related microservices and the three basic ones are mapped to the pseudocode of the Global Model as shown in Figure 4.6. The E.O. Service executes lines



```
1  $P \leftarrow$ GenerateInitialPopulation();
2  $P_1, P_2, ..., P_N \leftarrow$ Split($P$);
   // $N$ is the number of computing units (workers)
3  DistributeToWorkers($P_1, P_2, ..., P_N$);
4  foreach $worker\ j$ do
5      $EP_j \leftarrow$Evaluate($P_j$);
6  $EP \leftarrow$ Join($EP_{j1}, EP_{j2}, ..., EP_{jN}$);
7  while $not\ Termination\_Condition()$ do
8      $P' \leftarrow$ SelectParents($P$);
9      $P' \leftarrow$ ApplyVariationOperators($P'$);
10     $P'_1, P'_2, ..., P'_N \leftarrow$ Split($P'$);
11     DistributeToWorkers($P'_1, P'_2, ..., P'_N$);
12     foreach $worker\ j$ do
13         $EP'_j \leftarrow$Evaluate($P'_j$);
14     $EP' \leftarrow$ Join($EP'_{j1}, EP'_{j2}, ..., EP'_{jN}$);
15     $P_{next} \leftarrow$ SelectNewPopulation($P, P'$);
16 return $Best\ solution\ found$;
```

Evolutionary Operators Service
Distribution and Synchronization Service
Distribution and Synchronization Service
Calculation Service
Distribution and Synchronization Service
Evolutionary Operators Service
Distribution and Synchronization Service
Calculation Service
Distribution and Synchronization Service
Evolutionary Operators Service
Optimization Job Management Service

**Figure 4.6.:** Mapping the proposed microservices to the pseudocode of the Global Model

1,7, 8, 9 and 15, the Ds.S. Service implements lines 2, 3, 6, 10, 11 and 14, the Ca. Service performs the evaluation by executing lines 5 and 13, the Opt.J.M. Service corresponds to line 16.

Tables 4.12 and 4.13 summarize the REST-APIs and the publish/subscribe channels used by the services of the Global Model. While Table 4.12 shows the Sender (Sen.) and the Receiver (Rec.) that interact via each API, Table 4.13 defines the Publisher (Sub.) and the Subscriber (Sub.) for each channel.

| | | | **Microservices of the Global Model** | | | | | |
|---|---|---|---|---|---|---|---|---|
| **REST APIs** | **Method** | **UI** | **Opt.J. M.** | **E.O.** | **Ds.S.** | **Co. Ma.** | **Ca.** | **Da. Me.** |
| /opt/jobs | POST | Sen. | Rec. | | | | | |
| /opt/job/ {job-id}/start | POST | Sen. | Rec. | | | | | |
| /opt/job/ {job-id}/stop | POST | Sen. | Rec. | x | | | | |
| /opt/job/{job-id} | GET | Sen. | Rec. | | | | | |
| /opt/job/ {job-id}/results | GET | Sen. | Rec. | | | | | |
| /opt/job/{job-id} | DEL- ETE | Sen. | Rec. | | | | | |
| /com/ job/ {job-id}/create | POST | | Sen. | | | Rec. | | |
| /dam/job/{job-id}/ data/{data-type}/ store | POST | | Sen. | | | | | Rec. |
| /dam/job/{job-id} | GET | | Sen. | | | | | Rec. |
| /dam/job/{job-id}/ data/{data-type} | GET | | | | | | Sen. | Rec. |
| /eo/job/{job-id}/ configuration | PUT | | Sen. | Rec. | | | | |
| /eo/job/{job-id}/ status | GET | | Sen. | Rec. | | | | |
| /eo/job/{job-id}/ result | POST | | | Rec. | Sen. | | | |
| /ds/job/{job-id}/ configurations | POST | | Sen. | | Rec. | | | |
| /ds/job/{job-id}/ population | POST | | | Rec. | Sen. | | | |

**Table 4.12.:** Summary of the REST-APIs used by the services of the Global Model where (Rec.) refers to the receiver and (Sen.) refers to the sender

| | **Microservices of the Global Model** | | | | | |
|---|---|---|---|---|---|---|
| **publish/subscribe channels** | **Opt.J.M.** | **E.O.** | **Ds.S.** | **Co.Ma.** | **Ca.** | **Da.Me.** |
| status.service.{serviceName}.job.{job-id} | Sub. | Pub. | Pub. | Pub. | Pub. | Pub. |
| result.job.{job-id}.type.{resultType} | Sub. | | Pub. | | | |
| start.job.{job-id} | Pub. | Sub. | | | | |
| stop.job.{job-id} | Pub. | Sub. | Sub. | | Sub. | |
| population.job.{job-id}.worker.{worker-id} | | | Pub. | | Sub. | |
| config.job.{job-id}.worker.{worker-id} | | | Pub. | | Sub. | |
| result.part.job.{job-id}.worker.{worker-id} | | | Sub. | | Pub. | |

**Table 4.13.:** Summary of the publish/subscribe channels used by the services of the Global Model where (Pub.) refers to the publisher and (Sub.) refers to the subscriber

## 4.3. Mapping the Coarse-Grained Model to BeeNestOpt.IAI Architecture

In this section, we map the Coarse-Grained Model to the Container Layer of BeeNestOpt.IAI by using the three basic services, adapting some of the model-related services of the Global Model and adding new ones related to the Coarse-Grained Model. From the model-related services of the Global Model, some functionalities of the Ds.S. Service are used to support the Coarse-Grained Model. As explained in Section 2.2.1, the Coarse-Grained Model is more complex than the Global Model in terms of synchronization and has many parameters and options to be set up and distributed over the islands. Therefore, all functionalities related to the splitting the initial population, distributing them with configurations over the islands and joining the partial results are adapted and realized in a new service called Distributing & Joining Service (Ds.Jo. Service). The synchronization task is not more supported by the Ds.S. Service, since a new service called the Migration & Synchronization Service (Mi.Sy. Service) is designed to perform this task as we explain later. Besides these two services, the Initializer EA Service (In. EA Service) and the EA Service are designed. While the In. EA Service generates the initial population, the EA Service performs a sequential EA. The main tasks of the Coarse-Grained Model described in Section 2.2.1 are performed by these four new model-related services collaborating with the basic ones as shown in Figure 4.7.

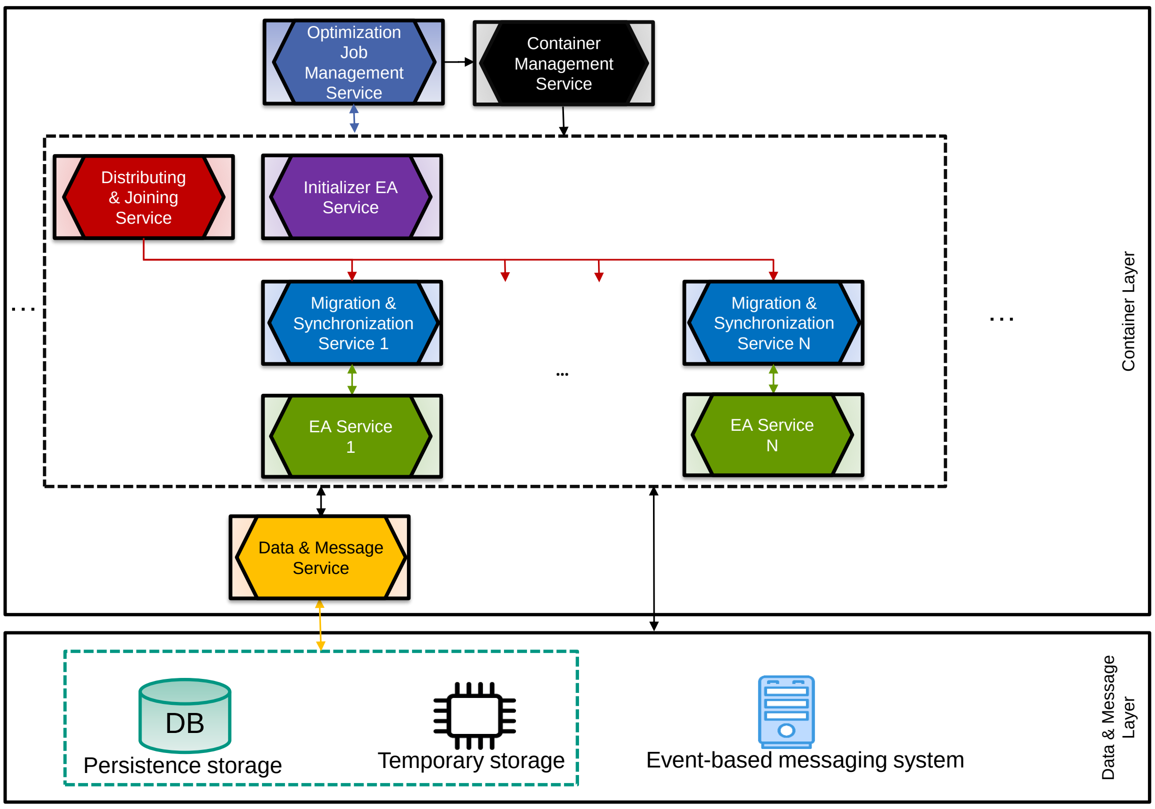

Optimization Job Management Service
Container Management Service
Distributing & Joining Service
Initializer EA Service
Migration & Synchronization Service 1
Migration & Synchronization Service N
EA Service 1
EA Service N
...
Data & Message Service
Container Layer
DB
Persistence storage
Temporary storage
Event-based messaging system
Data & Message Layer

**Figure 4.7.:** Mapping the Coarse-Grained Model to the cluster tier of BeeNestOpt.IAI

### 4.3.1. Coarse-Grained Model-related Services

In the following sections, we describe the newly added services in detail. For simplicity's sake, the term island refers to the Mi.Sy. Service and EA Service.

**Migration & Synchronization Service (Mi.Sy. Service)** The Migration & Synchronization Service is one of the model-related services for performing the Coarse-Grained Model in cluster computing environments. It is responsible for performing all tasks related to the migration policy, synchronizing the execution and checking if the global termination criterion is met within each island. Moreover, it builds the selected communication topology among the islands, by subscribing to the publish/subscribe channels of the respective neighbors as we explain later. The configurations related to the migration and communication topology are sent to each Mi.Sy. Service instance by the Ds.Jo. Service via a publish/subscribe messaging channel. Before an optimization job is started, each Mi.Sy. Service applies these configurations and publishes the status 'Initialized' to the status channel notifying the Opt.J.M. Service to start an optimization job. For exchanging migrants among islands, each Mi.Sy. Service publishes its migrants on its own channel which is subscribed by the neighbors. Defining which neighbors of an island subscribe to this channel is set up according to the selected communication topology. The Mi.Sy. Service supports synchronous migration policy among the islands where each Mi.Sy. Service subscribes to a shared channel, in which the status of the migration process is published. When the migration process is completed by all Mi.Sy. Service instances, they publish their status to the shared channel notifying other islands to start a new epoch of evolution. In addition, the Mi.Sy. Service supports asynchronous migration by ensuring that a migration process occurs in an asynchronous manner. In other words, each island starts a new epoch as soon as it receives the migrants from the considered neighbors and does not wait for the other islands to finish their migration process. The publish/subscribe channels used by each Mi.Sy. Service are described in Table 4.14.

**Distributing & Joining Service (Ds.Jo. Service)** The Distributing & Joining Service performs the same tasks of the Global Model related to splitting, distributing the initial population at the beginning of an optimization job and joining the results of the islands after finishing an optimization job or during an epoch for monitoring purposes. Furthermore, it distributes the EA, migration and communication topology configurations related to the Coarse-Grained Model over the islands before starting an optimization job. The distribution of configurations is subdivided into three steps. In the first step, the migration configuration is distributed. While the communication topology configuration is distributed in the second step, the EA configuration is sent in the last one. After the configuration is successfully distributed to the Mi.Sy. Service and EA Service instances, the Ds.Jo. Service notifies the Opt.J.M. Service via the status channel to call the In. EA Service for creating the initial population and sends it to the Ds.Jo. Service. The APIs of the Ds.S. Service listed in Table 4.8 are also used by the Ds.Jo. Service for receiving the configurations and the initial

| Method | Address | Description |
|---|---|---|
| PUBLISH | migrants.job.{job-id}. island.{island-id} | When an island finishes an epoch, it publishes its migrants to be consumed by the neighbors. |
| PUBLISH | stop.job.{job-id} | If an island reaches the global criterion, it notifies other islands by publishing a stop signal on this channel to stop their evolution process. |
| PUBLISH | status.migration.job. {job-id}.island. {island-id} | Each island continuously publishes the status of the migration process, i.e., 'Completed' or 'In process' which is used by applying the migration policy. |
| PUBLISH | start.EA.job. {job-id}.island. {island-id} | Publish a start signal to the corresponding EA Service to start the evolution process. |
| PUBLISH | result.part.job. {job-id}.island. {island-id} | Publish the partial result of an epoch. |
| SUBSCRIBE | config.migration.job. {job-id}.island. {island-id} | Get the migration configuration applied by a Mi.Sy. Service for an optimization job. |
| SUBSCRIBE | config.topology.job. {job-id} | Get the communication topology configuration, e.g., type of topology. |
| SUBSCRIBE | population.job. {job-id}.island. {island-id} | Get a subpopulation from the initial population assigned to this island. |
| SUBSCRIBE | population.intermediate. job.{job-id}.island. {island-id} | Get an intermediate population published by the EA Service after each epoch to select the migrants. |
| SUBSCRIBE | start.job.{job-id} | Get the start signal sent from Opt.J.M Service to start an optimization job. |
| SUBSCRIBE | migrants.job. {job-id}.island. {island-id} | Get the migrants from the neighbor islands defined by the applied communication topology. |
| SUBSCRIBE | stop.job.{job-id} | Get the stop signals from other Mi.Sy. Services instances. |
| SUBSCRIBE | status.migration.job. {job-id}.island. {island-id} | Get the status of the migration process. This is important for applying synchronous or asynchronous migration policy. |

**Table 4.14.:** The publish/subscribe channels of the Mi.Sy. Service for the Coarse-Grained Model

| Method | Address | Description |
|---|---|---|
| PUBLISH | config.EA.job.{job-id}. island.{island-id} | Publish the EA configuration for each EA Service. |
| PUBLISH | config.migration.job. {job-id}.island.{island-id} | Publish the migration configuration for each Mi.Sy. Service. |
| PUBLISH | config.topology.job. {job-id} | Publish the communication topology configuration applied by all Mi.Sy. Service. |
| PUBLISH | population.job.{job-id}. island.{island-id} | Publish subpopulations obtained from the initial population for each island. |
| PUBLISH | result.job.{job-id}.type. {resultType} | Publish the joined intermediate or final results of an optimization job. |
| SUBSCRIBE | result.part.job.{job-id}. island.{island-id} | Get the partial result related to each island. |

**Table 4.15.:** The publish/subscribe channels of the Ds.Jo. Service for the Coarse-Grained Model

population related to an optimization job. The publish/subscribe channels used by each Ds.Jo. Service are described in Table 4.15

**EA Service** For each island, there is an EA Service instance which is called by the Mi.Sy. Service to evolve the population. The same design concept followed by designing the E.O. Service in the Global Model is also considered for the EA Service. In other words, it can be designed as a standalone microservice or encapsulated in the I/O adapter as shown in Figure 4.5. Compared to the Global Model, the EA Service performs the tasks of the E.O. Service and the Ca. Service. This includes the parent selection, the application of genetic operators, the acceptance of the offspring and the evaluation of the offspring internally by calculating the objective functions and constraints. After finishing each epoch, the EA Service sends the intermediate population to the corresponding Mi.Sy. Service to perform the migration process according to the migration policy. After performing it, the Mi.Sy. Service publishes the newly obtained population into the corresponding channel of the island. The EA Service reads it from the channel and uses it as an initial population. The EA Service implements only the Event-based communication style enabling more flexibility by building different communication topologies. The publish/subscribe channels used by each EA. Service are described in Table 4.16.

**Initializer EA Service (In. EA Service)** The main purpose of the In. EA Service is to create an initial population according to a certain configuration sent by the Opt.J.M. Service for an optimization job. The initial population is distributed over the islands at the beginning of each optimization job. The initial population can be created completely or partly randomly. In the latter case, some individuals are taken from previous optimization jobs or from an external tool. It has two APIs as shown in Table 4.17

| Method | Address | Description |
|---|---|---|
| PUBLISH | population.intermediate. job.{job-id}.island. {island-id} | Publish an intermediate population after each epoch to be handled by the corresponding Mi.Sy. Service. |
| SUBSCRIBE | config.EA.job.{job-id}. island.{island-id} | Get an EA specific configuration from Ds.Jo. Service. |
| SUBSCRIBE | start.EA.job.{job-id}. island.{island-id} | Get the start signal from the Mi.Sy. Service to start the evolution process. |
| SUBSCRIBE | stop.job.{job-id} | Get the stop signal from the corresponding Mi.Sy. Service when the global criterion is reached. |
| SUBSCRIBE | config.EA.job.{job-id}. island.{island-id} | Get the EA configuration to be applied by EA Service for an optimization job. |

**Table 4.16.:** The publish/subscribe channels of the EA Service for the Coarse-Grained Model

| Method | URL-Pattern | Description |
|---|---|---|
| PUT | /ine/job/{job-id}/ configuration | A PUT request on this URL is used to set up the EA configuration of the In. EA Service. This can include, e.g., the used chromosome model. |
| POST | /ine/job/{job-id}/ population | A POST request on this URL is used to generate an initial population according to the applied EA configuration. |

**Table 4.17.:** The APIs of the In. EA Service for the Coarse-Grained Model

### 4.3.2. Adaptation of the Basic Services to Support the Coarse-Grained Model

For the basic services, no new features have been added to the Co.Ma. Service and the Da.Me. Service. However, the functionalities of the Co.Ma. Service is adapted to handle the request of creating the number of containers for the Mi.Sy. Service and the EA Service instances and other services, namely the Ds.Jo. Service and the In. EA Service. The Co.Ma. Service assigns an identifier (island-id) to the Mi.Sy. Service and the EA Service that form an island. This identifier is used to facilitate exchanging data among the islands themselves and between them and other services.

The Opt.J.M. Service performs the basic functionalities described in Section 4.2.2, i.e., sending a request to the Co.Ma. Service for creating the required instances of the Mi.Sy. Service and the EA Service and other services. It is also responsible for starting and stopping an optimization job. A new function to create the initial population by calling the In. EA Service is added to the Opt.J.M. Service. In addition to that, the Opt.J.M. handles the new configurations related to the Coarse-Grained Model which can be categorized into two groups. While the first group includes the migration configuration, namely the migration rate, migration frequency, migrant selection and migrant replacement policy, the second one comprises the type of the selected communication topology, e.g., a uni-directional ring (Ring), a bi-directional ring (Bi Ring), a ladder (Ladder) and a completely connected graph (Complete) and information on how to apply them, i.e, static or dynamic. Static means that only one communication topology is applied during the evolution process of an optimization job. If the topology changes during one optimization job then we obtain the dynamic topology. Having these configurations and the EA configuration increase the flexibility of BeeNestOpt.IAI enabling the deployment of homogeneous and heterogeneous islands where each island receives separate migration and EA configuration. Furthermore, it supports the deployment of a static as well as a dynamic Coarse-Grained Model.

Since the Coarse-Grained Model applies a complex communication model among islands, the Event-based messaging system of the Data & Message Layer is used to realize the publish/subscribe pattern as an underlying communication paradigm. This ensures a full decoupling between the services and facilitates the establishment of different communication topologies. For example, to realize a bi-directional ring topology, the Mi.Sy. Service with ID $i$ publishes the migrants to its own publish channel $i$ and the neighboring Mi.Sy. Service instances with ID $i+1$ and $i-1$ subscribe to this channel. In the example depicted in Figure 4.8, the island 3 publishes migrants to its channel (No. 3). In the example depicted in Figure 4.8, the island 3 publishes migrants to its channel (No. 3). The islands 2 and 4, as neighbors and subscribers, subscribe the channel No. 3 to get the published migrants.

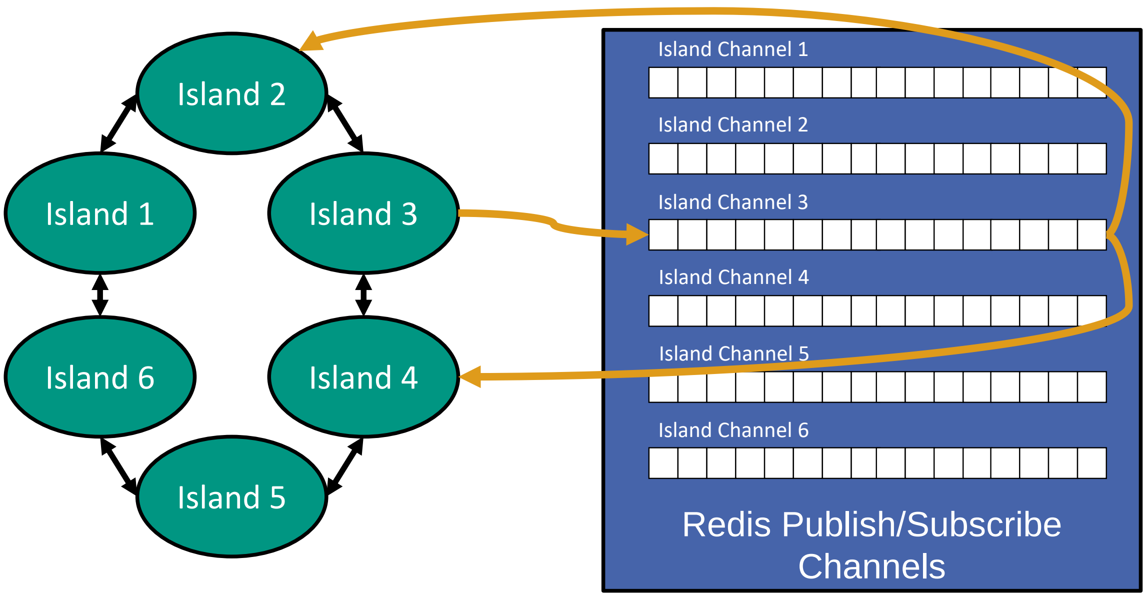


**Figure 4.8.:** Mapping a bi-directional ring topology with 6 islands to the publish/subscribe channels infrastructure

### 4.3.3. Execution Workflow

For the execution of the Coarse-Grained Model, the three general phases proposed in BeeNestOpt.IAI namely the ‘Initialization Phase’, the ‘Iterative Evolution Phase’ and the ‘Termination Phase’ are also performed. In comparison to the execution workflow of the Global Model explained in Section 4.2.3, the execution workflow of the Coarse-Grained Model is more complex. Therefore, we simplify the explanation of the phases as much as possible for better understanding.

**Initialization Phase** After performing the general ‘Initialization Phase’ by creating the required containers for the model-related services, i.e, the In. EA Service, the Ds.Jo. Service, the Mi.Sy. Service and the EA Service instances, they subscribe to the associated publish/subscribe channels. After that, the initial population is created by calling the In. EA Service by the Opt.J.M. Service. Then, the Opt.J.M. Service starts to send the initial population and configurations to the Ds.Jo. Service which in turn splits the initial population into subpopulations and distributes them over the islands. The initial configurations are also distributed to the islands. It is possible that the Mi.Sy. Service instances receive a complete, partial or no initial subpopulation at all. In the latter two cases, each EA Service has to generate either the missing part of the received subpopulation if it has received a smaller initial subpopulation than the required one, or a complete subpopulation if it did not receive any initial subpopulation. The individuals generated in this process are randomly generated. The initial configurations include the configurations for the deployed EA, migration strategy parameters, e.g., migration rate, selection and replacement strategy and the communication topology configuration which is required for building the communication topology among the islands. The Mi.Sy. Service instances can subscribe to their neighbors in several manners defined by the user which leads to establishing different topologies like a Ring, Bi Ring, Ladder or Complete as we explained before. In case of a heterogeneous Coarse-Grained Model, a list with all configurations related to each island is sent by the Ds.Jo. Service to all islands, where each island takes its configurations from the list according to the island-id. After finishing the Initialization Phase, the Mi.Sy. Service and the EA Service instances have all the required data to execute the evolution by, e.g., applying the genetic operators, evaluating the individuals and exchanging migrants.

**Iterative Evolution Phase** After finishing the ‘Initialization Phase’, each Mi.Sy. Service informs the Opt.J.M. Service that it is ready to execute the actual iterative cycle of the evolution. Once the Opt.J.M. Service receives an ‘Initialized’ status from all Mi.Sy. Service instances and a start signal from UI Tier, a start command is sent to the islands to start the evolution. In the ‘Iterative Evolution Phase’, each Mi.Sy. Service starts its corresponding EA Service that evolves the population by evaluating the individuals, selecting the parents and applying the genetic operators, to name a few. The evolution process is performed iteratively until some epoch termination criterion has been satisfied. Afterward, each Mi.Sy. Service receives the intermediate population from the corresponding EA Service to select the migrants according to the selection policy. Then, the migrants are published on their corresponding publish/subscribe channel as shown in Figure 4.8. The neighbors, i.e., the Mi.Sy. Service instances receive migrants through the same channel and replace individuals from their intermediate population according to the replacement strategy. After that, the EA Service is called by its corresponding Mi.Sy. Service with the updated population including the migrants. This loop is performed as long as there is no global termination criterion satisfied.

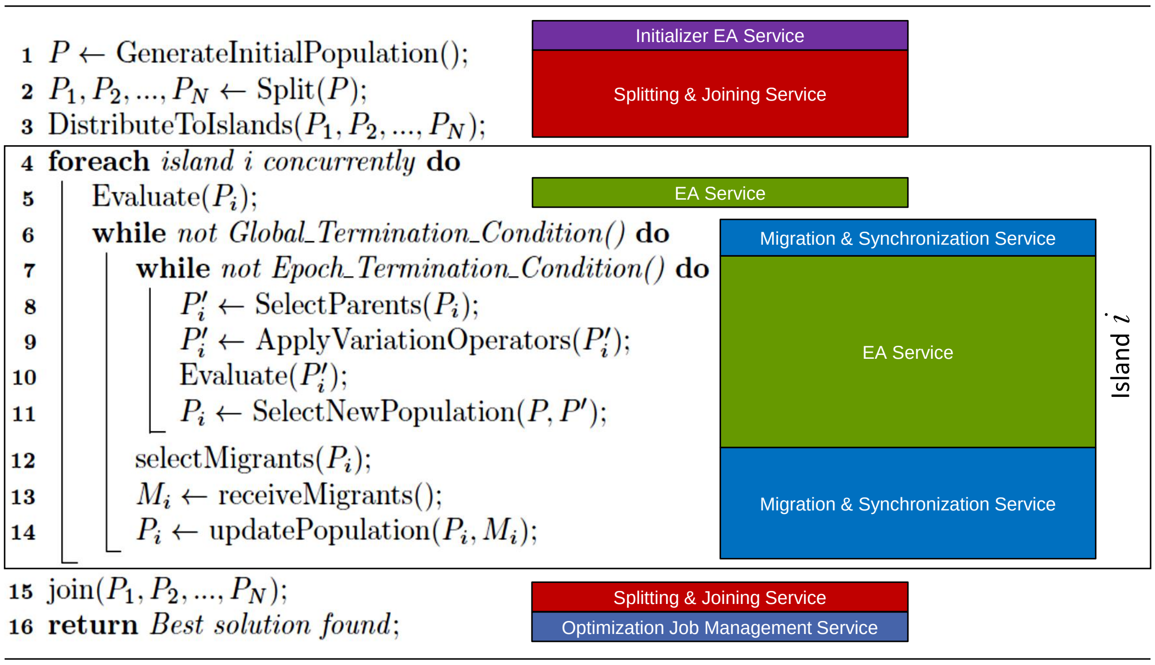
**Termination Phase** In the final phase of an optimization task, either one Mi.Sy. Service notifies the other Mi.Sy. Service instances to stop the evolution process or all Mi.Sy. Service instances stop themselves. The distinction is caused by the fact that there are multiple termination criteria. For example, if the termination criterion is the number of generations or the computation time, the Mi.Sy. Service instances stop themselves, since there is no knowledge necessary from the other Mi.Sy. Service instances to make this decision. However, if the termination criterion is the minimum fitness of a potential solution, one Mi.Sy. Service notifies the others that it has found a solution that exceeds the required fitness value. Hence, this information is published to all Mi.Sy. Service instances in order to finish their evolving process. After that, the Mi.Sy. Service instances send a stop signal to the Opt.J.M. Service to stop the evolution process. The Mi.Sy. Service instances publish their partial results to the corresponding publish/subscribe channels. The Ds.Jo. Service selects the best obtained results, after being notified that the partial results are available to be joined in one list. The steps of the general 'Termination Phase' explained in Section 4.1.2 is also performed. Table 4.18 gives an overview how the above microservices are involved in the three stages of the execution workflow.

| Services | Initialization Phase | Iterative Evolution Phase | Termination Phase |
|---|---|---|---|
| Opt.J.M. Service | Yes | No | Yes |
| In. EA Service | Yes | No | No |
| Ds.Jo. Service | Yes | Yes | Yes |
| Co.Ma. Service | Yes | No | No |
| Mi.Sy. Service | Yes | Yes | Yes |
| EA Service | Yes | Yes | Yes |

**Table 4.18.:** Mapping the microservices of the Coarse-Grained Model to the three execution phases

The proposed microservices are mapped to the pseudocode of the Coarse-Grained Model (see, Figure 4.9). The In. EA Service executes line 1, the Ds.Jo. Service executes lines 2, 3 and 15, the Mi.Sy. Service implements lines 6 and 12-14, the EA Service corresponds to lines 5 and 7-11 and the Opt.J.M. Service returns the best solution by performing line 16.

Besides the REST-APIs supported by the Opt.J.M. Service and the Co.Ma. Service of the Global Model summarized in Table 4.12, Table 4.19 contains the REST-APIs of the Ds.Jo. Service and the In. EA Service for supporting the Coarse-Grained Model. It also defines the Sender (Sen.) and the Receiver (Rec.) of each API. Table 4.20 lists all publish/subscribe



```
1  $P \leftarrow$ GenerateInitialPopulation();
2  $P_1, P_2, ..., P_N \leftarrow$ Split($P$);
3  DistributeToIslands($P_1, P_2, ..., P_N$);
4  foreach island i concurrently do
5      Evaluate($P_i$);
6      while not Global_Termination_Condition() do
7          while not Epoch_Termination_Condition() do
8              $P'_i \leftarrow$ SelectParents($P_i$);
9              $P'_i \leftarrow$ ApplyVariationOperators($P'_i$);
10             Evaluate($P'_i$);
11             $P_i \leftarrow$ SelectNewPopulation($P, P'$);
12         selectMigrants($P_i$);
13         $M_i \leftarrow$ receiveMigrants();
14         $P_i \leftarrow$ updatePopulation($P_i, M_i$);
15 join($P_1, P_2, ..., P_N$);
16 return Best solution found;
```

Initializer EA Service
Splitting & Joining Service
EA Service
Migration & Synchronization Service
EA Service
Migration & Synchronization Service
Island i
Splitting & Joining Service
Optimization Job Management Service

**Figure 4.9.:** Mapping the proposed microservices to the pseudocode of the Coarse-Grained Model

| | | Microservices of the Coarse-Grained Model | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **REST APIs** | **Method** | **Opt.J. M.** | **EA** | **Ds.Jo.** | **Co. Ma.** | **Mi.Sy.** | **Da. Me.** | **In. EA** |
| /ds/job/{job-id}/ configurations | POST | Sen. | | Rec. | | | | |
| /ds/job/{job-id}/ population | POST | Sen. | | Rec. | | | | |
| /ine/job/{job-id}/ configuration | PUT | Sen. | | | | | | Rec. |
| /ine/job/{job-id}/ population | POST | Sen. | | | | | | Rec. |

**Table 4.19.:** Summary of the REST-APIs used by the services of the Coarse-Grained Model where (Rec.) refers to the receiver and (Sen.) refers to the sender

channels used by the services of the Coarse-Grained Model and shows which service is the Publisher (Pub.) and which is the Subscriber (Sub.) for each channel.

| | **Microservices of the Coarse-Grained Model** | | | | | | |
|---|---|---|---|---|---|---|---|
| **publish/subscribe channels** | **Opt.J. M.** | **EA** | **Ds.Jo.** | **Co. Ma.** | **Mi.Sy.** | **Da. Me.** | **In. EA** |
| status.service. {serviceName}.job. {job-id} | Sub. | Pub. | Pub. | Pub. | Pub. | Pub. | Pub. |
| migrants.job.{job-id}. island.{island-id} | | | | | Pub. (Sub. by neighbors) | | |
| stop.job.{job-id} | Pub. | Sub. | | | Pub. (Sub. by neighbors) | | |
| status.migration.job. {job-id}.island{island-id} | | | | | Pub. (Sub. by neighbors) | | |
| start.EA.job.{job-id}. island.{island-id} | | Sub. | | | Pub. | | |
| result.part.job.{job-id}. island.{island-id} | | | Sub. | | Pub. | | |
| config.migration.job. {job-id}.{island-id} | | | Pub. | | Sub. | | |
| config.topology.job. {job-id} | | | Pub. | | Sub. | | |
| population.job.{job-id}. island.{island-id} | | | Sub. | | Sub. | | |
| population.intermediate. job.{job-id}.island. {island-id} | | Pub. | | | Sub. | | |
| start.job.{job-id} | Pub. | | | | Sub. | | |
| migrants.job.{job-id}. island.{island-id} | | | | | Sub. | | |
| status.migration.job. {job-id}. island{island-id} | | | | | Pub. (Sub. by neighbors) | | |
| config.EA.job.{job-id}. island.{island-id} | | Sub. | Pub. | | | | |
| result.job.{job-id}.type. {resultType} | Sub. | | Pub. | | | | |

**Table 4.20.:** Summary of the publish/subscribe channels used by the services of the Coarse-Grained Model where (Pub.) refers to the publisher and (Sub.) refers to the subscriber

## 4.4. Evaluation

In this section, we investigate how BeeNestOpt.IAI facilitates the deployment of existing EAs that already support at minimum one parallelization model from the basic ones into a cluster computing environment without any change or adaptation at the code level. Moreover, the effect of using microservices, container virtualization and the publish/subscribe messaging paradigm on the parallel performance of an EA in the cluster computing environment is studied and analyzed. Basically, the parallel performance of EAs can be measured and compared in two ways. Either one measures the effort required to achieve a given and attainable target quality (speedup) or one measures the quality of solutions achieved at a given effort. In other words, parallel EAs can be evaluated with respect to two criteria, namely how fast can an EA obtain an optimal or near-optimal solution in comparison to the sequential version of it (speedup) and what is the quality of this solution (solution quality). In this evaluation, we focus on the first criterion, namely the speedup. Moreover, the performance of BeeNestOpt.IAI in terms of the communication overhead is evaluated.

There are two different approaches to analyze the speedup, namely the theoretical analysis (e.g., [36]) and the experimental analysis (e.g., [122]). In this evaluation, we analyze the performance of BeeNestOpt.IAI empirically following [122]. The speedup ($S_m$) is defined as the ratio between the sequential and parallel execution times using $m$ processors. Therefore, it is necessary to define how execution time is measured. In the sequential case, it is the CPU time to solve the problem, since the algorithm is executed on a single processor. In the parallel case, the execution time has to include the communication overhead arising from exchanging data between the services. Consequently, we measure the time for executing the parallel EA by taking into account the start and the end time – known as wall-clock times – for each optimization task subtracting the communication overhead. We denote $T_m$ as the execution time of a parallel EA on $m$ processors and $T_1$ as the execution of the same EA on one processor as shown in Equation (4.1).

$$s_m = \frac{T_1}{T_m} \tag{4.1}$$

Because of the non-deterministic nature of EAs, we cannot use this metric directly. Therefore, we have to consider the means for several runs $E[T_1]$ and $E[T_m]$ for both measures as described in Equation (4.2).

$$s_m = \frac{E[T_1]}{E[T_m]} \tag{4.2}$$

Driven by this definition, we can distinguish between three cases: Sublinear speedup with ($s_m < m$), linear speedup with ($s_m = m$) and superlinear speedup with ($s_m > m$). Alba [2] introduced a taxonomy depending on the definition of these values (see, Table 4.21).

Strong speedup (type I.) refers to the speedup achieved compared to the best-so-far sequential algorithm. Because of the difficulty to find the best-so-far algorithm for a problem at hand, most of the research works do not consider strong speedup. Instead, the weak speedup (type II.) which refers to a comparison between the sequential version and the parallel version

| |
|---|
| I. Strong speedup |
| II. Weak speedup |
| A. speedup with solution-stop |
| 1. Versus panmixia |
| 2. Orthodox |
| B. speedup with predefined effort |

**Table 4.21.:** Taxonomy of speedup Measures taken from [2]

of the developed EA is applied. The two subcategories of the weak are distinguished from each other based on the stopping criterion. For speedup with solution-stop (type II.A), a target solution quality is given. The parallel version of an EA running on *m* processors is either compared to the sequential version, i.e., the panmictic EA running on one processor (type II.A.1) or compared to the same parallel EA running on one processor (type II.A.2). The second subcategory sets a predefined effort as the stopping criterion (type II.B.). This is especially useful if not the performance of a given EA should be evaluated but the parallel environment it is executed in. In this evaluation, we consider type II.B. speedup as the most suitable measure.

In the following, we describe the EA integrated into BeeNestOpt.IAI as a test case. Afterward, we introduce the deployment steps on the cluster to parallelize EAs. The obtained benchmark results of the evaluation of he Global Model and the Coarse-Grained Model in terms of the communication overhead and speedup are discussed.

### 4.4.1. EA GLEAM as Test Case

The EA GLEAM (General Learning Evolutionary Algorithm and Method) [23, 24] shortly explained in Section 2.2.3 is chosen as a test case valuating the Global Model and Coarse-Grained Models realized in the microservices and container-based architecture. However, any EA can be integrated in BeeNestOpt.IAI as long as the proposed interfaces are implemented which facilitates the processing of new applications. The variant of GLEAM used in this work offers the functionality to configure a scalable delay so that the amount of time for the fitness evaluation can be artificially extended. While the EA GLEAM is integrated into the E.O. Service and acting as a master in the Global Model, it is integrated into two services in the Coarse-Grained Model, namely the In. EA Service and the EA Service. The communication interfaces of the EA-related services provide a GLEAM-specific layer that allows the integration of GLEAM as it is. The Fine-Grained Model supported by GLEAM is not considered in this evaluation, since it needs a large massively parallel processing infrastructure to assign each individual to a CPU which is not available in our cluster computing environments.

https://github.com/KIT-IAI/Gleam, last visited: 13/01/2023

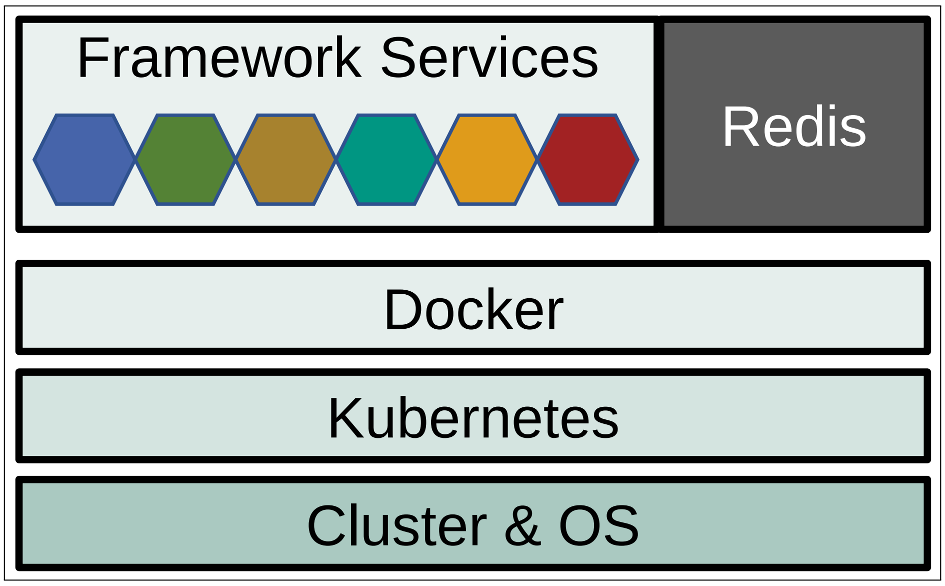


**Figure 4.10.:** The technological layers of BeeNestOpt.IAI

### 4.4.2. Deployment and Execution on a Cluster

Figure 4.10 gives an overview about the involved technological layers of BeeNestOpt.IAI. On the lowest level, the cluster hardware runs a certain Operating System (OS). However, all microservices run in a containerized environment that is independent of the underlying OS. Therefore, we do not describe the OS in detail. On top of the OS runs the container orchestration system, namely Kubernetes which dynamically allocates resources to Pods that run the microservices. Inside the Pods, the implemented microservices are run by the Docker Engine. In addition to the implemented services, Redis runs in a Pod acting as a temporary storage and implementation for the Event-based messaging system. For this evaluation, we use a cluster with four nodes with 32 Intel cores (2.4 GHz), 128 GB RAM and an SSD disk each. The nodes are connected to each other by a LAN with 10 GBit/s bandwidth.

As a starting point, we have the source code for all microservices that perform the Global Model and the Coarse-Grained Model. Some of these services are developed in Python and the others in Java. For the latter case, each service has besides the source code the Maven's Project Object Model (POM) file which describes the project dependencies and configuration details. Combining both files together creates the executable Java Archive (JAR) for each microservice which is encapsulated in a local docker image as shown in Figure 4.11. In order to deploy the developed microservices on a cluster, two parts are necessary. The first one is the docker images which are pushed to an online docker registry. The second part is the YAML config which describes the number of CPUs and size of RAM assigned for each docker, the name of the deployment, how many Pods for one deployment will be created, networking details and other relevant information for Kubernetes. This information is included into cluster configuration provided as a JSON file defined by a user. For the services that do not scale on demand, one Pod containing one container for each

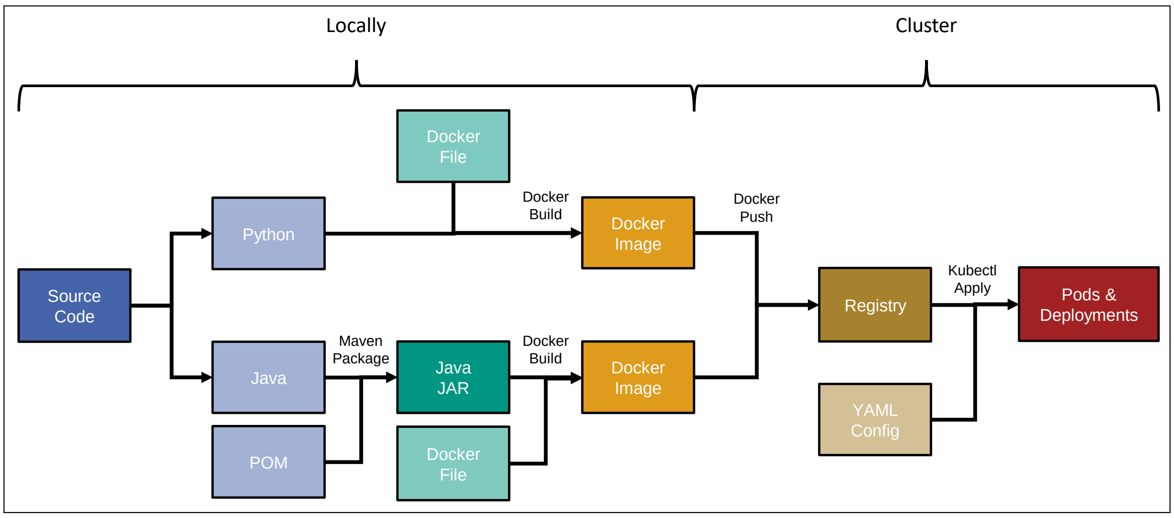


**Figure 4.11.:** The deployment process of BeeNestOpt.IAI

microservices is created. However, for creating several instances from the other services, several Pods are created where each one contains one container running one microservices such as the Ca. Service, the Mi.Sy. Service and the EA Service. Hence, the terms container and Pod refer to the same component in this evaluation. Figure 4.12 shows an example of how the services of the Coarse-Grained Model with four islands are mapped to the CPUs on four cluster nodes. On the first node, the Opt.J.M. Service, the Ds.Jo. Service and the In. EA Service are deployed. The second node runs the Co.Ma. Service, the Mi.Sy. Service 1 as well as the EA Service 1. Node number three runs the Mi.Sy. Service instances 2, the EA Service 2 and the EA Service 3. On the fourth node, Redis, the Mi.Sy. Service instances 3 and 4, the EA Service 4 and the Da.Me. Service are running. Since Kubernetes automatically allocates resources depending on the resource usage, the distribution of the microservices can change between two succeeding optimization tasks.

## 4.4.3. Experimental Setup and Results for the Global Model

For estimating the applicability and the efficiency of the proposed microservice and container-based architecture for parallelizing EAs according to the Global Model on a cluster, we analyze the scalability of the Global Model. This is achieved by varying the number of workers between 1 and 120, so that each worker becomes one core and at minimum two cores are left on every node for the operating system. Since the development of the overhead with growing data volume and amount of workers is of particular interest for the Global Model, we vary the population size between 120 and 960 individuals. The optimization problem which is used as a test case is a scheduling task from the industrial process [86, 23] where each chromosome consists of 87 genes each with 2 integer parameters. This results in chromosome list sizes of 820 kB and 6560 kB for 120 and 960 individuals, respectively. Each run is performed for 50 generations so that all overhead-related functions are performed 50 times to minimize the effect of single delays coming from the operating

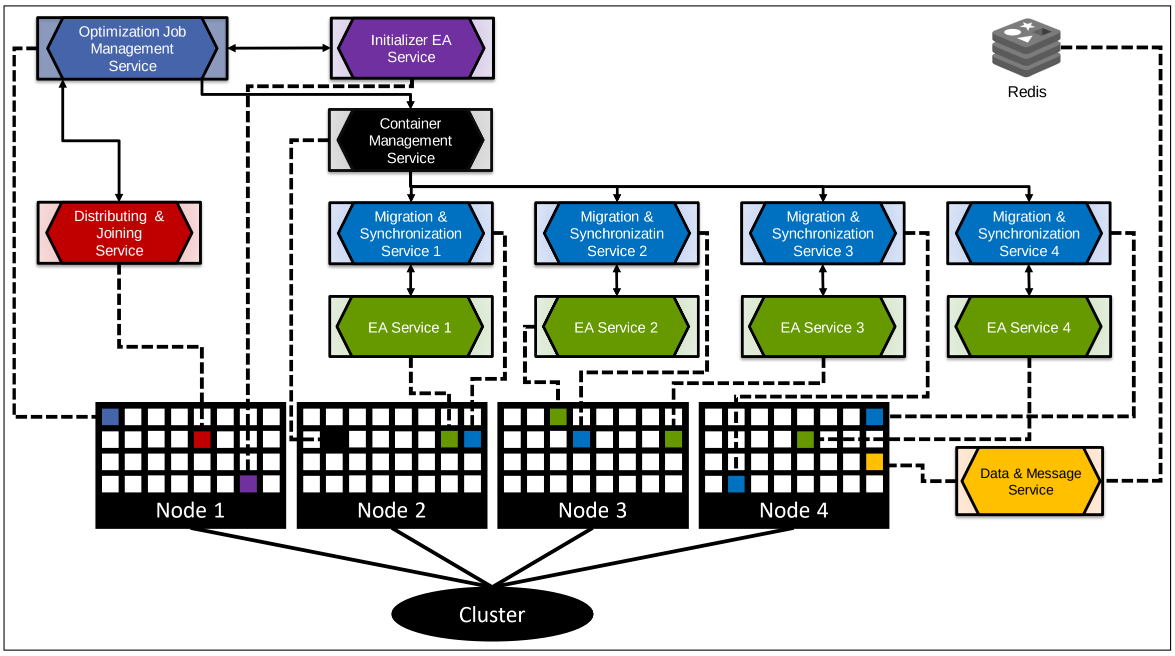

Optimization Job Management Service
Initializer EA Service
Redis
Container Management Service
Distributing & Joining Service
Migration & Synchronization Service 1
Migration & Synchronizatin Service 2
Migration & Synchronization Service 3
Migration & Synchronization Service 4
EA Service 1
EA Service 2
EA Service 3
EA Service 4
Node 1
Node 2
Node 3
Node 4
Data & Message Service
Cluster

**Figure 4.12.:** Example of how the containers of BeeNestOpt.IAI for the Coarse-Grained Model with 4 islands are distributed over a cluster with 4 nodes

system. The total time and the maximum time required to evaluate the allotted portion of the offspring of all Ca. Services are measured. The maximum time is taken, since the Ds.S. Service must wait for the last Ca. Service to finish, i.e, we consider the synchronous Global Model. The scalable delay feature of the used version of EA GLEAM for the fitness evaluation is applied for simulating different run times. Figure 4.13 shows a nearly linear increase of the overhead time with growing population sizes ($\mu$) averaged over the number of involved workers, i.e., the Ca. Service instances. The growth is originated from the linear increase in data volume exchanged among the services and the increasing effort required for managing and synchronizing the workers as it is depicted in Figure 4.14. In other words, by deploying more workers, an additional overhead arises, since BeeNestOpt.IAI requires more time to orchestrate the execution over the workers. The concluded result shows that BeeNestOpt.IAI with the Global Model generates an overhead of only 0.4–1.8 seconds per generation for up to 120 worker nodes and chromosome lists of size up to 6.5 MB.

Table 4.22 and Figure 4.15 show the results of the scalability experiment, for which the execution time of one evaluation of an individual is artificially set to 0.9 seconds. The results show that by utilizing more workers, the gain in the execution time has an almost linear trend between 8 and 32 workers. Then, it increases more slowly between 32 and 120 workers. This is due to the fact that the master becomes a bottleneck when using too many workers, since most of them try to communicate with the master to exchange data at the same time. Cantú-Paz analyzed among others the performance of the Global Model [36] and defined the ratio $\gamma$ between the time required for one assessment ($t_{eval}$) and the communication overhead ($t_{com}$) as shown in Equation (4.3).

$$\gamma = \frac{t_{eval}}{t_{com}} \tag{4.3}$$

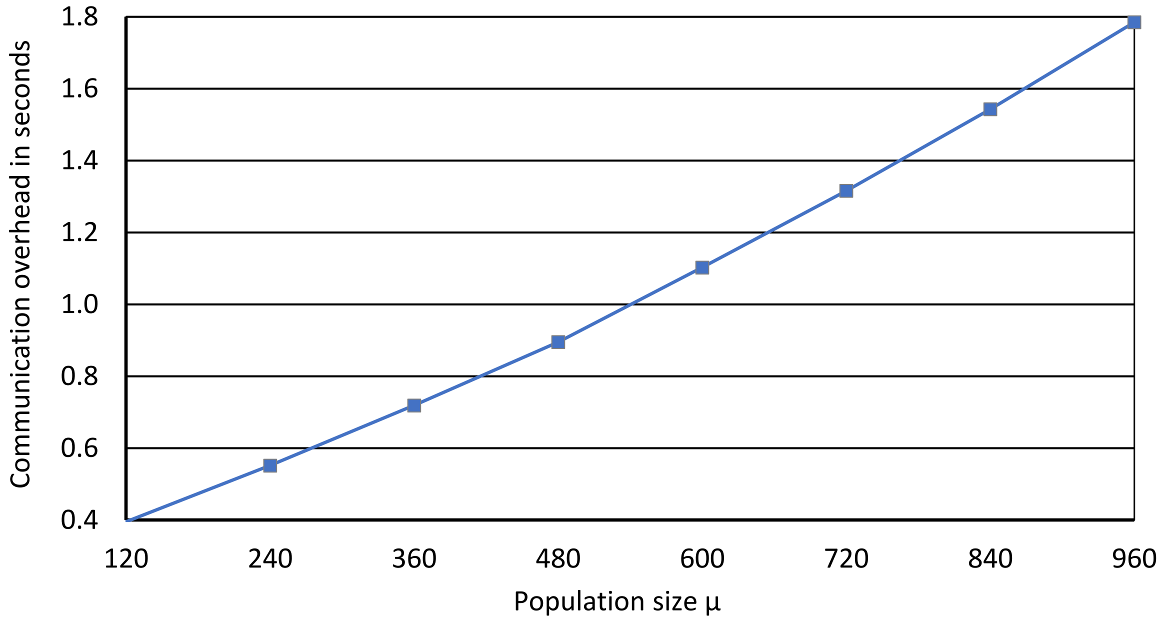


**Figure 4.13.:** Overhead times with increasing population size $\mu$ per generation as average of the workers (ranged between 8 and 120)

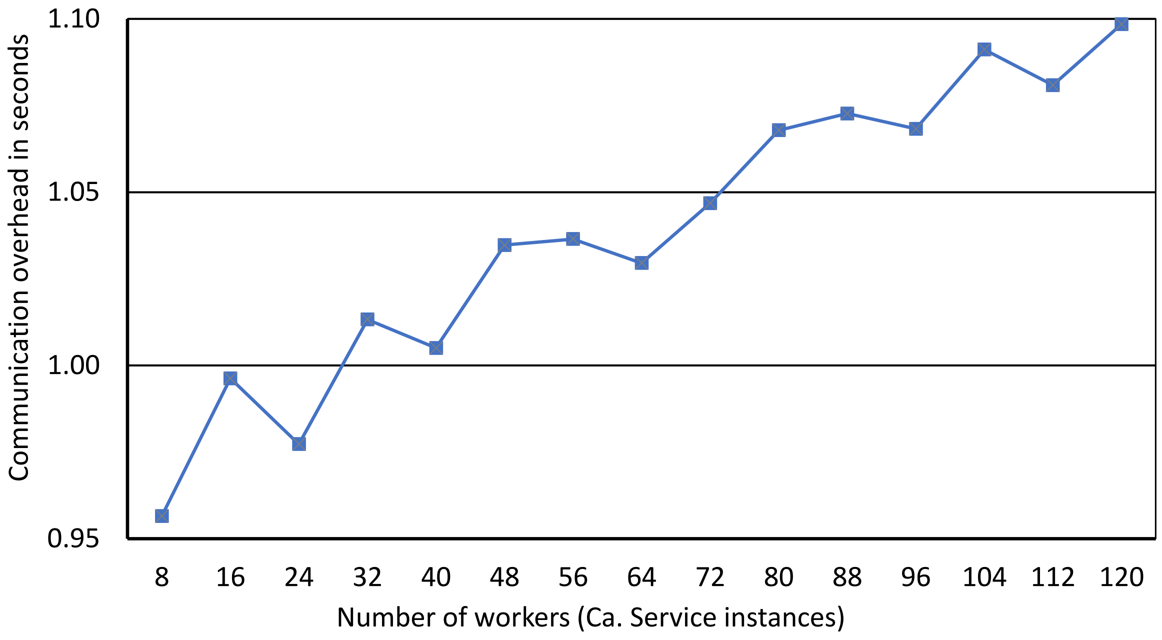


**Figure 4.14.:** Overhead times with increasing number of workers per generation as average of the population sizes (ranged between 120 and 960)

The value of $\gamma$ should be greater or even better much greater than one. Given that value, the number of optimal workers $S^*$ can be calculated for the given test case, in which the master node does not also work as a worker, as shown in Equation (4.4).

$$S^* = \sqrt{\lambda\gamma} = \sqrt{8\mu\gamma} \tag{4.4}$$

| **Containers [Workers]** | **Execution Time [Seconds]** | **Overhead [Seconds]** | **Total Time [Seconds]** | **Speedup** |
|---|---|---|---|---|
| 1 | 87216.42 | 25.73 | 87242.15 | 1 |
| 8 | 12110.80 | 24.68 | 12135.47 | 7.19 |
| 16 | 5883.40 | 23.84 | 5907.24 | 14.77 |
| 24 | 4220.87 | 24.74 | 4245.62 | 20.55 |
| 32 | 3240.90 | 24.98 | 3265.89 | 26.71 |
| 40 | 2713.06 | 24.84 | 2737.90 | 31.86 |
| 48 | 2336.97 | 25.42 | 2362.39 | 36.93 |
| 56 | 1978.08 | 25.97 | 2004.04 | 43.53 |
| 64 | 1838.47 | 25.56 | 1864.03 | 46.80 |
| 72 | 1768.33 | 26.27 | 1794.60 | 48.61 |
| 80 | 1686.85 | 26.71 | 1713.56 | 50.91 |
| 88 | 1621.74 | 26.69 | 1648.44 | 52.92 |
| 96 | 1551.67 | 26.98 | 1578.64 | 55.26 |
| 104 | 1516.49 | 26.40 | 1542.88 | 56.54 |
| 112 | 1485.98 | 26.12 | 1512.10 | 57.70 |
| 120 | 1423.01 | 27.67 | 1450.68 | 60.14 |

**Table 4.22.:** Performance of BeeNestOpt.IAI with μ=240 and 50 generations per run

where $\lambda$ is the amount of offspring per generation which is set to $8 * \mu$ (the EA GLEAM is set to generate eight offspring per paring). From table 4.22, we can calculate the $t_{eval}$ per individual according to the following equation:

$$t_{eval} = \frac{execution\_time\_sequential}{\mu + (8 * \mu * Nr.ofGenerations)} \tag{4.5}$$

and $t_{com}$ as following:

$$t_{com} = \frac{overhead}{Nr.ofGenerations} \tag{4.6}$$

For an execution time of 0.9 seconds for one evaluation of an individual and $\mu$=240, the optimal number of workers is 58 and $\gamma = 1.76$. This corresponds to the curve shown in Figure 4.15, the increase of which starts to decrease beginning at approx. 50–60 workers. This low and only slightly rising overhead time allows an efficient parallelization starting at an assessment duration of approx. 0.1 seconds per individual. For this duration, the best number of workers ranges between 15 and 40 for the population sizes used.

### 4.4.4. Experimental Setup and Results for the Coarse-Grained Model

For assessing the performance of BeeNestOpt.IAI for parallelizing EAs based on the Coarse-Grained Model, we study the overhead including the communication overhead in each

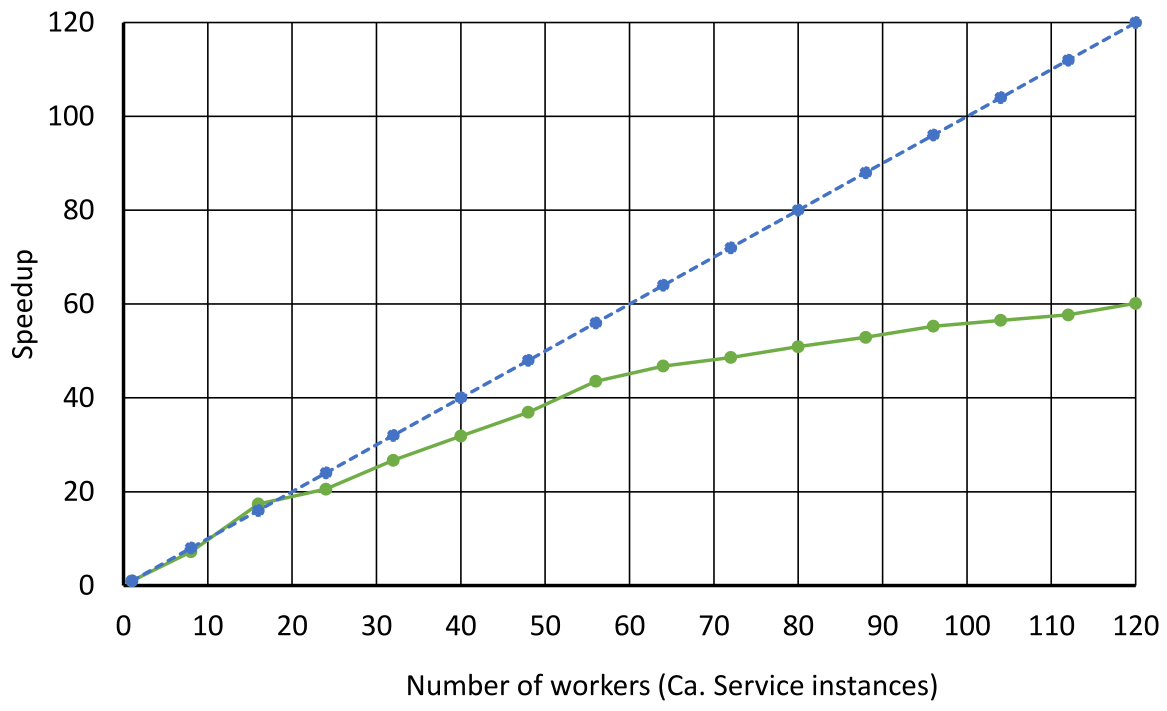


**Figure 4.15.:** Theoretical speedup (dotted line) vs. BeeNestOpt.IAI speedup for the Global Model

phase of the execution workflow and the migration overhead. Furthermore, the parallel performance in terms of speedup is analyzed. To this end, the non-convex non-linear multimodal Rastrigin function [152] is selected in its generalized form as an optimization problem. The only relevant information for the purpose of this evaluation is the chosen dimensionality of the function of 20, so that an individual contains 20 genes requires a size of about 2 KB. For simulating a real scenario, in which the computing resources are exploited as they would be in a real optimization task, we defined several evaluation scenarios for evaluating the overhead and the speedup of BeeNest32.IAI for the Coarse-Grained Model as we explain in the following sections. In these scenarios, we compare four communication topologies, namely a Ring, a Bi Ring, a Ladder and a Complete.

#### 4.4.4.1. Overhead

For measuring the communication and the migration overhead of the microservice and container-based architecture for parallelizing EAs according to the Coarse-Grained Model, we change the following two parameters:

- Number of islands: it is varied between 1, 8, 16, 32, 64 and 120, so that the minimum of two cores is left on each node for the OS.
- Migration rate: we chose migration rates between 1 (the absolute minimum of the migration rate), 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192 and 16384.

According to these migration rates, the amount of exchanged data per island and epoch ranged between 2 KB and 32 MB. For each migration rate, we execute 100 epochs so that the overhead related functions are calculated 100 times to minimize the effect of single delays resulting from the OS or other running services on the cluster. As a starting point to analyze the overhead, we measure and compare the amount of time needed for starting up the required islands. The Co.Ma. Service dynamically deploys the required number of containers by either creating them, if there are no deployed ones or resetting them. The latter is applied, if they are already deployed and are not used by another optimization task. For both cases, there is a linear increase of time with the number of islands as shown in Figure 4.16. However, the amount of time needed to create a container is about 20 times greater than to reset it. Due to the varied load on the cluster, some minor fluctuations can be observed such as the ones between 40 and 56 containers.

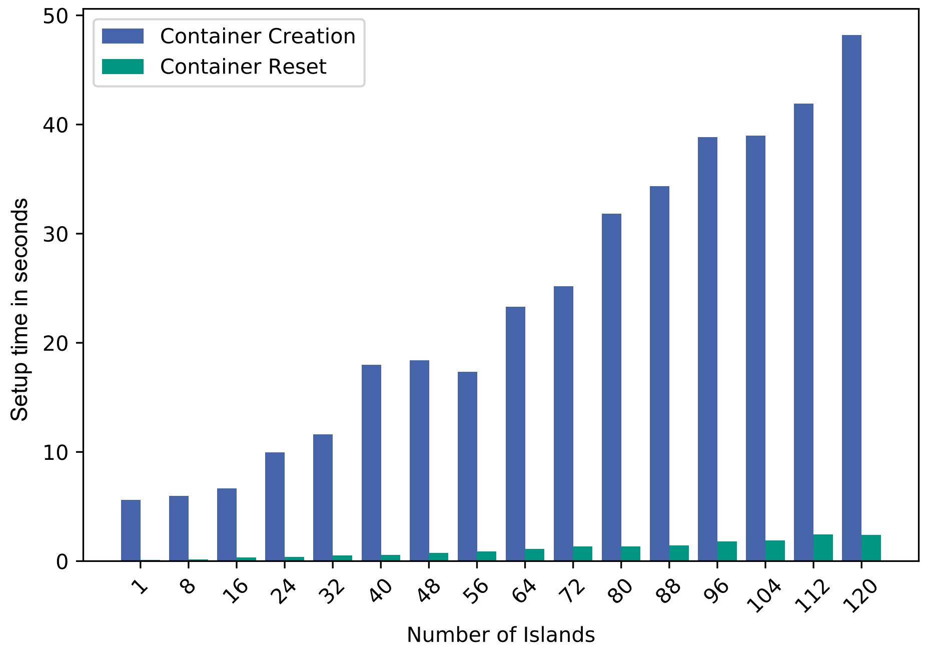


**Figure 4.16.:** Comparison of the setup time for the islands for a complete container creation and a reset

After presenting the setup time, we focus on the BeeNestOpt.IAI overhead of all phases of the execution workflow, namely the ‘Initialization Phase’ (with a container reset), the ‘Iterative Evolution Phase’ (excluding the computation time for the deployed EA in the EA Service) and the ‘Termination Phase’. As described in Section 4.3.3, the ‘Initialization Phase’ and the ‘Termination Phase’ are executed only once for each optimization task. Therefore, we add up both and compare it to the amount of time needed for the ‘Iterative Evolution Phase’.

As shown in Figure 4.17, the overhead of the ‘Initialization Phase’ and the ‘Termination Phase’ for Ring topology with 8 islands remains almost constant when migration rates vary between 1 and 1024. In contrast to that, the overhead of the ‘Iterative Evolution Phase’ is accountable for the increase of the overhead starting with 256 migrants. Therefore, we focus on the overhead resulting from the ‘Iterative Evolution Phase’. In this phase, the active

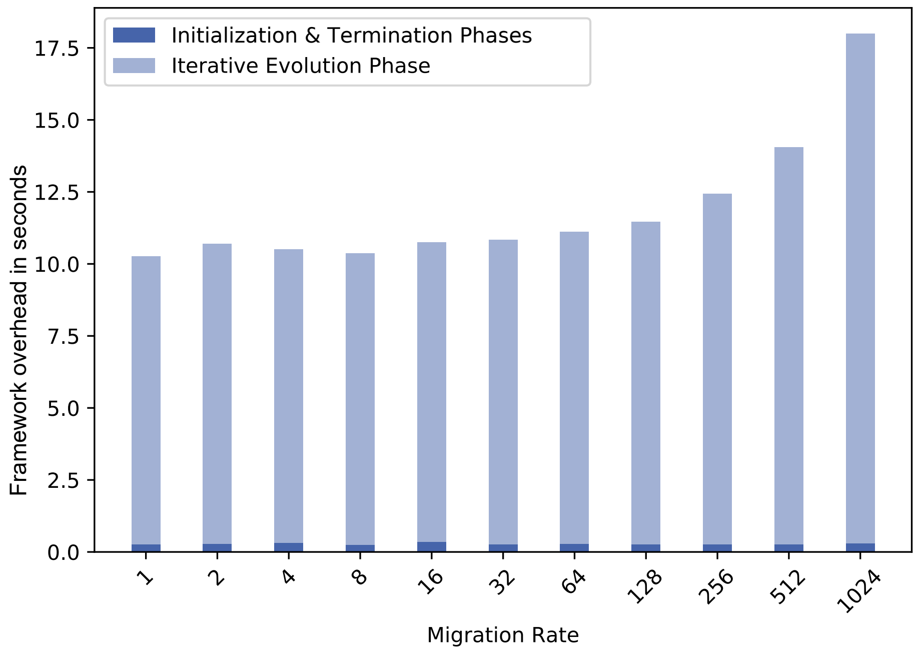


**Figure 4.17.:** Framework overhead for a Ring topology with 8 islands and 100 epochs, spitted into overhead for the 'Initialization & Termination Phases' and 'Iterative Evolution Phase'

services are the Mi.Sy. Service and the EA Service. We summarize both as an island which executes $N$ epochs. The time of each epoch is subdivided into the amount of time needed to execute the deployed EA (GLEAM Execution Time in Figure 4.18) and the amount of time needed by BeeNestOpt.IAI to perform the migration (Framework Migration Overhead (FMO) in Figure 4.18).

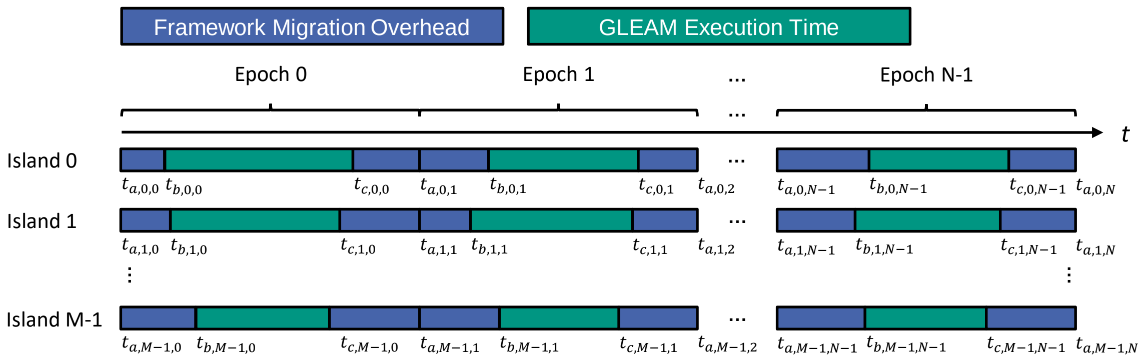


**Figure 4.18.:** Points in Time of the 'Iterative Evolution Phase' of one optimization task performed by $M$ islands for $N$ epochs

For different combinations of parameters, only the FMO is considered, since it is the governing factor of the overhead. For one optimization task, we observe the FMO for each island and each epoch building the migration overhead matrix $s$ as shown in Equation (4.7).

$$s = \begin{pmatrix} s_{0,0} & s_{0,1} & s_{0,2} & \cdots & s_{0,N-1} \\ s_{1,0} & s_{1,1} & s_{1,2} & \cdots & s_{1,N-1} \\ s_{2,0} & s_{2,1} & s_{2,2} & \cdots & s_{2,N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ s_{M-1,0} & s_{M-1,1} & s_{M-1,2} & \cdots & s_{M-1,N-1} \end{pmatrix} \tag{4.7}$$

Then, this matrix is converted to the vector $t$ as shown in Equation (4.8) containing the FMO for each epoch by calculating the maximum FMO of each epoch from all islands as shown in Equation (4.9). The maximum FMO of all islands is considered, since it gives us the worst case of the FMO.

$$t = \begin{pmatrix} t_0 & t_1 & t_2 & \cdots & t_{N-1} \end{pmatrix} \tag{4.8}$$

$$\underset{0 \le j \le N-1}{t_j} = \max_{i \in \{0,..,M-1\}} s_{i,j} \tag{4.9}$$

The sum of the maximum FMO, namely $r$ for $N$ epochs is shown in Equation (4.10). The whole procedure to calculate the FMO for one optimization task is summarized in Algorithm 1.

$$r = \sum_{j=0}^{N-1} t_j \tag{4.10}$$

We compare the FMO per epoch for the Ring, Bi Ring, Ladder and Complete topologies with 8 islands. Figure 4.19 shows the results for varying the migration rate between 1 and 1024. It is noticeable that the FMO rises by increasing the number of connections between the islands, e.g., the FMO for the Bi Ring, Ladder and Complete topologies is higher than for the Ring. This is mainly due to the fact that the number of exchanged messages among the islands has increased. We increase the migration rate from 1024 to 2048, 4096, 8192 and 16384. The default limit of each channel, i.e., 32 MB per 60 seconds is not enough to handle the amount of exchanged data. Therefore, the output buffer has to be increased to be able to handle more than 8192 migrants with 8 islands. As shown in Figure 4.20, the migration overhead trend shown in Figure 4.19 is also confirmed.

**Algorithm 1:** Pseudocode to calculate the Migration Overhead (FMO) for one optimization task based on the points in time shown in 4.18

```
input  : All times related to one optimization task as shown in Figure 4.18
output : FMO ($r$) for one optimization task
1  $M$ = Number of islands;
2  $N$ = Number of epochs;
3  $s$ = [M,N], matrix with migration overhead times for each island and epoch (see,
     Equation (4.7));
4  $t$ = [N], vector with maximum migration overhead for each epoch (see, Equation (4.8)),
     each entry initialized with $-\infty$;
5  $r$ = Migration overhead for one optimization task (see, Equation (4.10));
6  for $i \leftarrow 0$ to $M - 1$ do
7      for $j \leftarrow 0$ to $N - 1$ do
8          $s_{i,j} = (t_{a,i,j+1} - t_{c,i,j}) + (t_{b,i,j} - t_{a,i,j})$;
9  for $j \leftarrow 0$ to $N - 1$ do
10     for $i \leftarrow 0$ to $M - 1$ do
11         $t_j = max(t_j, s_{i,j})$;
12 for $j \leftarrow 0$ to $N - 1$ do
13     $r = r + t_j$;
14 return $r$;
```

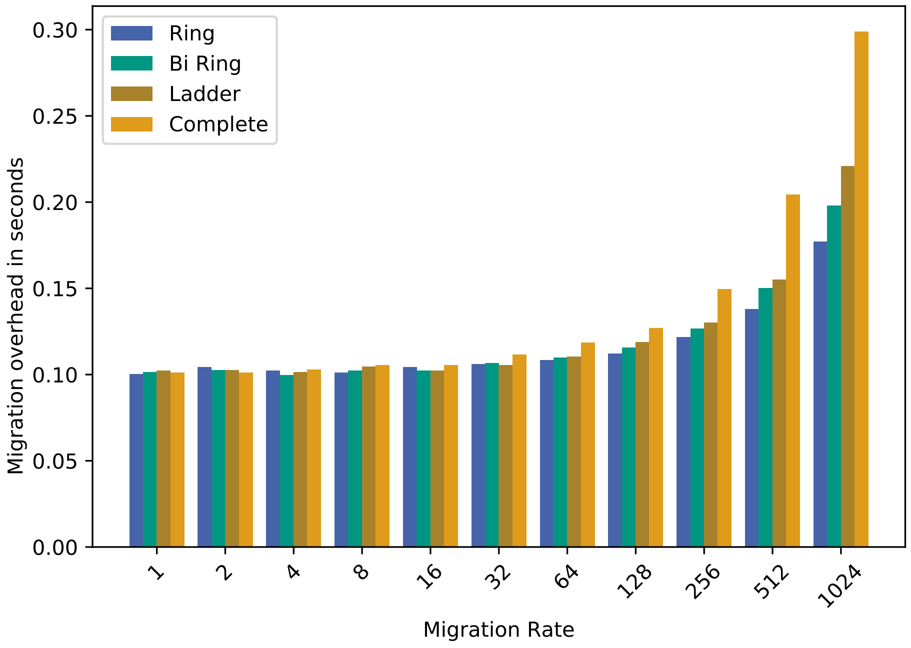


**Figure 4.19.:** Migration overhead per epoch for a Ring, Bi Ring, Ladder and Complete with 8 islands by varying the migration rate between 1 and 1024, smaller part of (4.20)

After comparing the influence of different topologies and migration rates on the FMO, we focus on the influence of the number of islands. The migration rate is varied between 1,

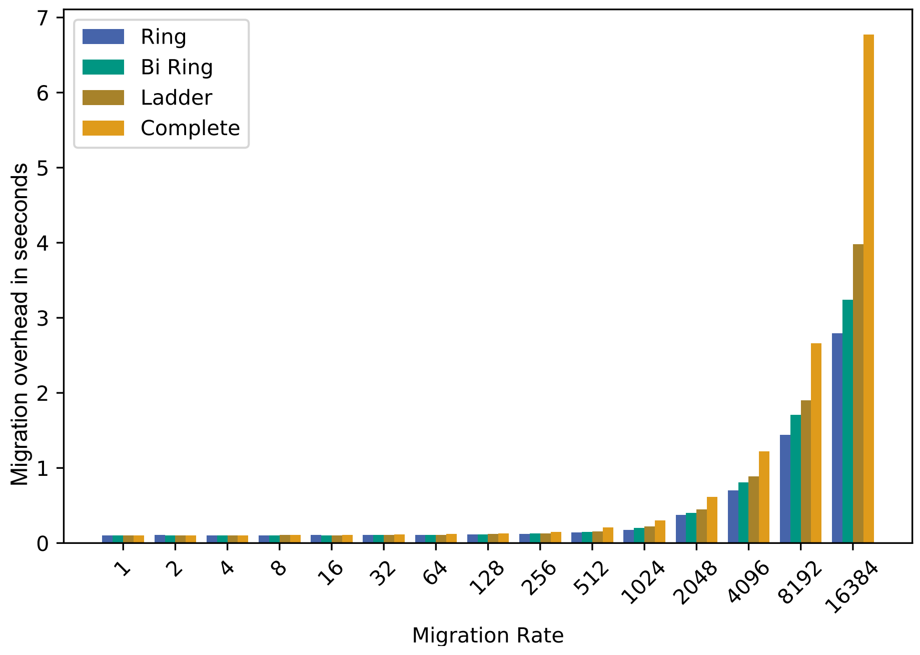


**Figure 4.20.:** Migration overhead per epoch for a Ring, Bi Ring, Ladder and Complete graph topology with 8 islands varying the migration rate between 1 and 16384

2, 4, 8, 16, 32, 64 and 128. For 8 islands (see, Figure 4.19) the considered topologies and migration rates show an almost constant FMO. For 16 islands (see, Figure 4.21), we can see an increase of the FMO for Complete topology starting with a migration rate of 32. For 32 islands (see, Figure 4.22), the increase of the FMO for the complete graph topology also starts at a migration rate of 32. For 64 islands (see, Figure 4.23), the Complete topology shows peaks by 16 migrants and for 120 islands (see, Figure 4.24), peaks begin at a migration rate of 8. The reason for these peaks is the exponential growth of connections with a growing number of islands, especially for Complete topology. For the Ring, Bi Ring and Ladder topologies, we always have a constant number of neighbors of 1, 2 and 3, respectively.

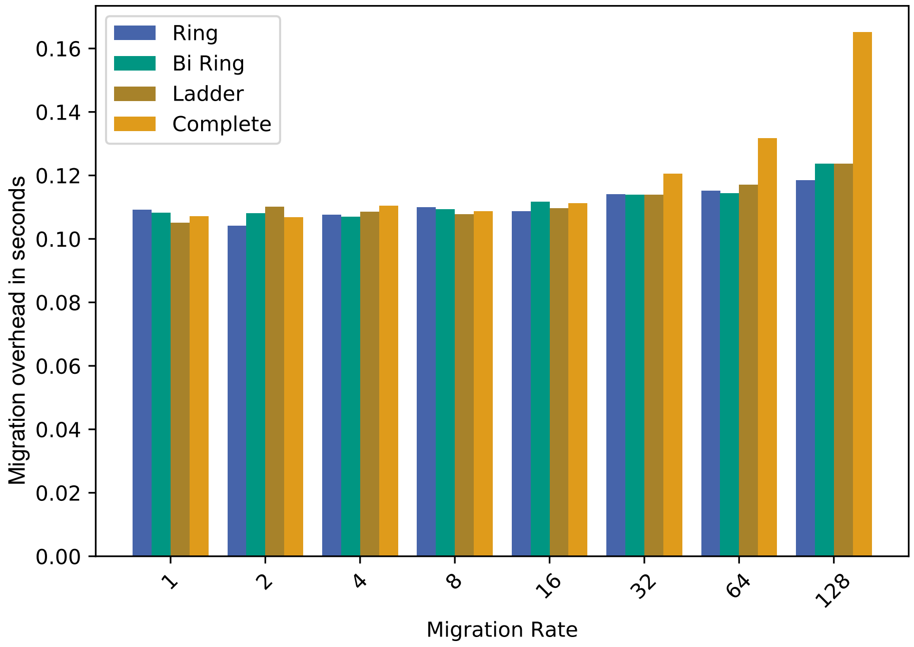


**Figure 4.21.:** Migration overhead per epoch for a Ring, Bi Ring, Ladder and Complete graph topology with 16 islands

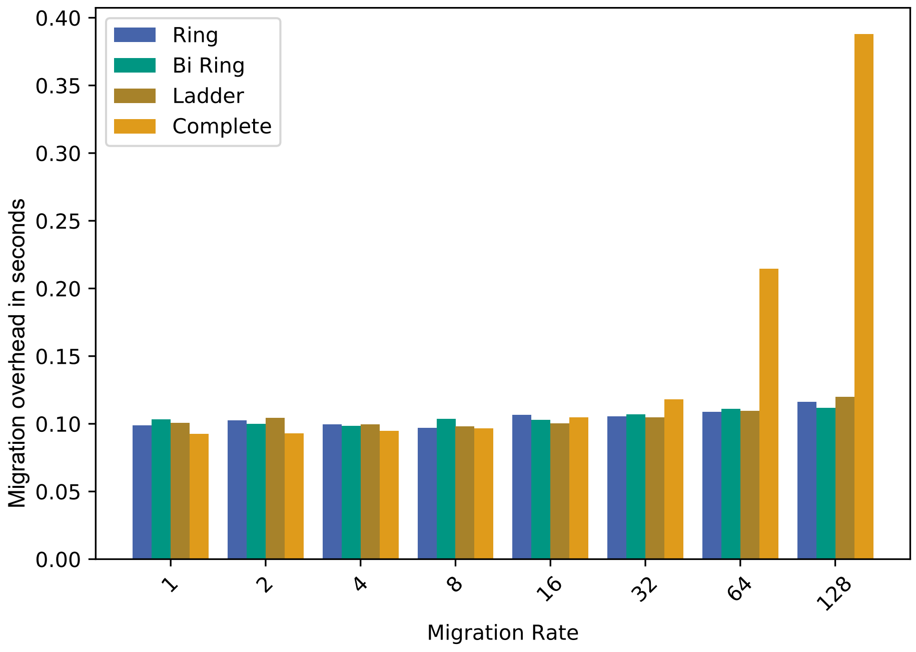


**Figure 4.22.:** Migration overhead per epoch for a Ring, Bi Ring, Ladder and Complete graph topology with 32 islands

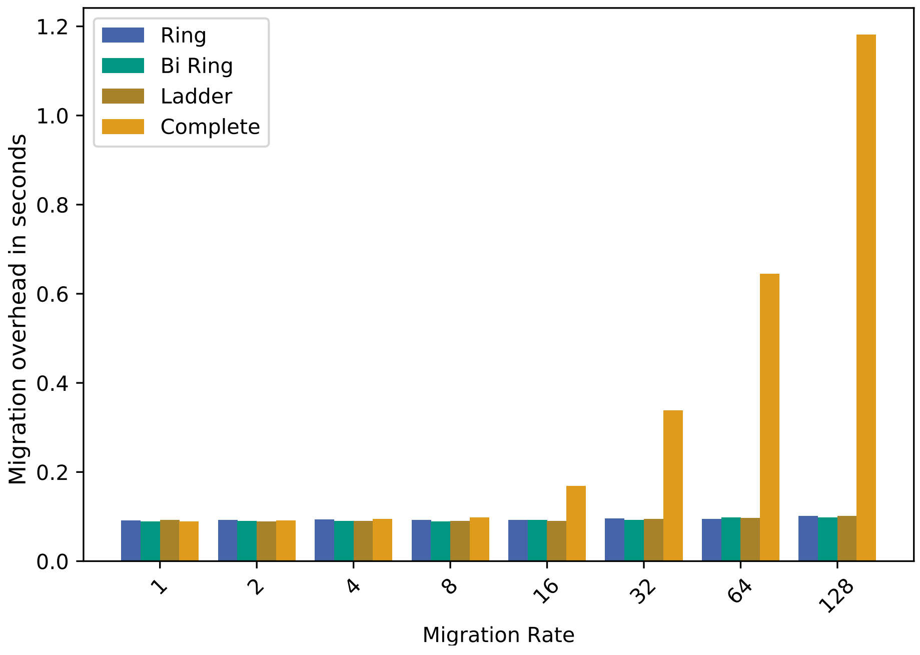


**Figure 4.23.:** Migration overhead per epoch for a Ring, Bi Ring, Ladder and Complete graph topology with 64 islands

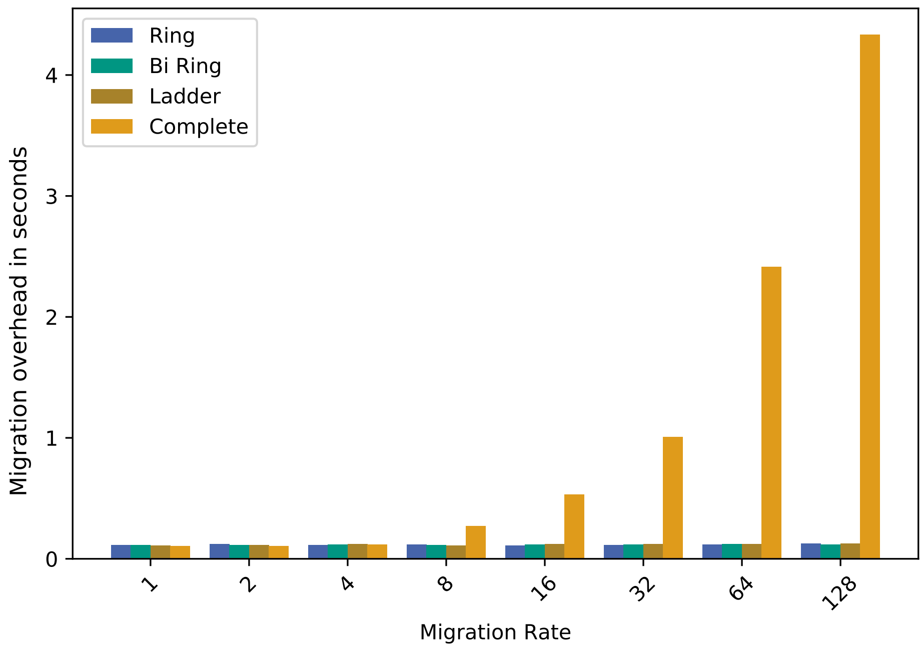


**Figure 4.24.:** Migration overhead per epoch for a Ring, Bi Ring, Ladder and Complete graph topology with 120 islands

#### 4.4.4.2. Speedup

For measuring the speedup of the parallel EAs according to the Coarse-Grained Model and based on the microservices and container technologies, the delay feature of the applied GLEAM version for the fitness evaluation explained in Section 2.2.3 is used. We vary the delay between 0 ms (just executing the optimization for the Rastrigin function), 1 ms, 2 ms, 4 ms, 8 ms, 16 ms and 32 ms to simulate optimization tasks with several levels of complexity. In the parallel execution, the global population size is set to 1024 and the migration rate to 4 for the four considered topologies. The migration among the islands is synchronously occurring and all islands have the same configurations, i.e., they are homogeneous. The number of islands is varied between 8, 16, 24, 32, 40, 48, 56, 64, 72, 80, 88, 96, 104, 112 and 120. By dividing the whole population, i.e., 1024 among the islands, subpopulation sizes ranging between 9 and 128 individuals are obtained. The number of epochs (the interval between two succeeding migrations) is set to 100 with 1 generation per epoch as a termination criterion. For the sequential implementation, we execute the EA GLEAM on one island with 100 epochs where each epoch performs one generation. The population consists of 1024 individuals resulting in the same effort (number of evaluations of the fitness function) as of the parallel EA. We measure the whole execution time of BeeNestOpt.IAI, i.e., between the reception of the configurations sent from the UI tier in the Opt.J.M. Service and the reception of the final result in the Opt.J.M. Service. Then, the speedup is averaged over 10 runs in order to minimize the possible effect of the delays caused by the OS and the non-deterministic behavior of EAs.

The results for the speedup measured for the Ring, Bi Ring, Ladder and Complete topologies are shown in Figures 4.25, 4.26, 4.27 and 4.28, respectively. By increasing the delay, the speedup shows a partial linear/superlinear trend for all four topologies. This is due to the fact that the computation time of the sequential EA increases rapidly by increasing the delay. In contrast to this linear increase, the computation time of the parallel EA increases more slowly because of the distribution of computation over the islands. Moreover, the migration overhead is independent of the delay and therefore the ratio between the overhead and the parallel computation time decreases. This results in a higher speedup for greater delays. There are three possible reasons introduced by Alba [2] for this partial superlinear behavior of parallel EA based on the Coarse-Grained model, namely the implementation source, numerical source and the physical source. In the implementation source, the underlying data structure used to represent the chromosomes has a large impact on the reduced execution time of the parallel implementation. For instance, the computing units for a Structured EA handle shorter lists of individuals other than a sequential one. On the other hand, the complexity of certain operators, e.g., selection is significantly reduced when the population is split and the resulting subpopulations are dealt with in parallel. In the numerical source, the parallel EA can find a solution of the same quality faster as the one found by the sequential EA by preserving the genetic diversity for a longer time by searching many regions of the search space concurrently. The last source, i.e., the physical one refers to the capability of efficiently exploiting the deployed computing resources by applying a parallel version of EA more than by applying a sequential EA.

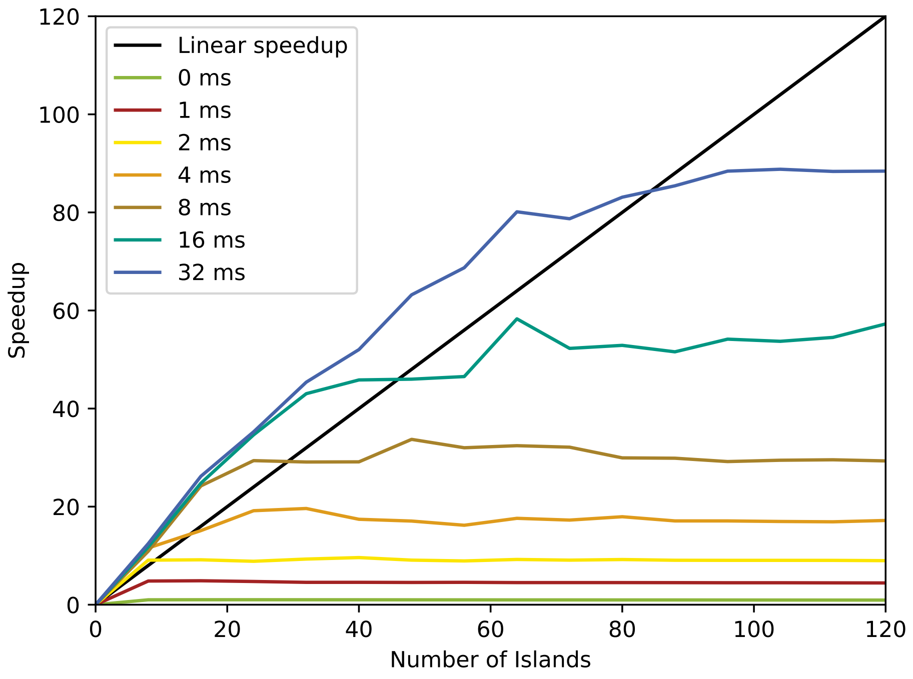


**Figure 4.25.:** Speedup with 4 migrants and a Ring topology, delay varied between 0 and 32 ms

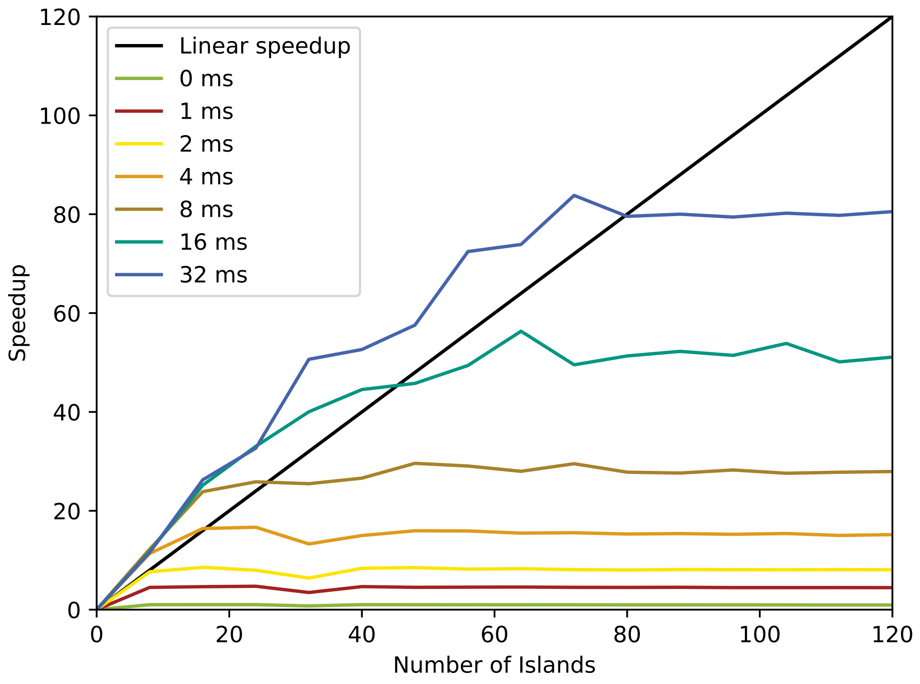


**Figure 4.26.:** Speedup with 4 migrants and a Bi Ring topology, delay varied between 0 and 32 ms

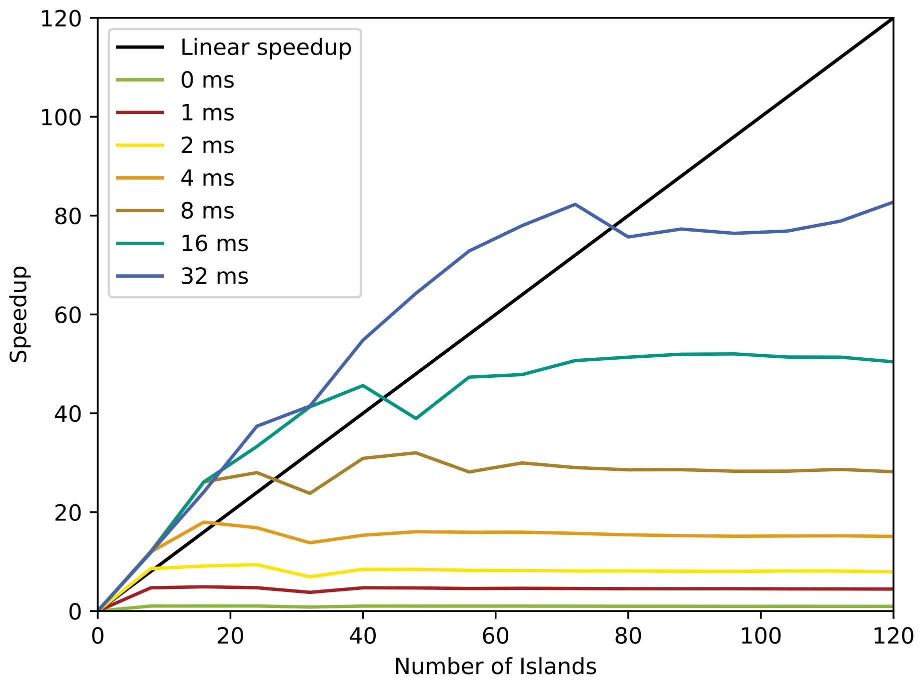


**Figure 4.27.:** Speedup with 4 migrants and a Ladder topology, delay varied between 0 and 32 ms

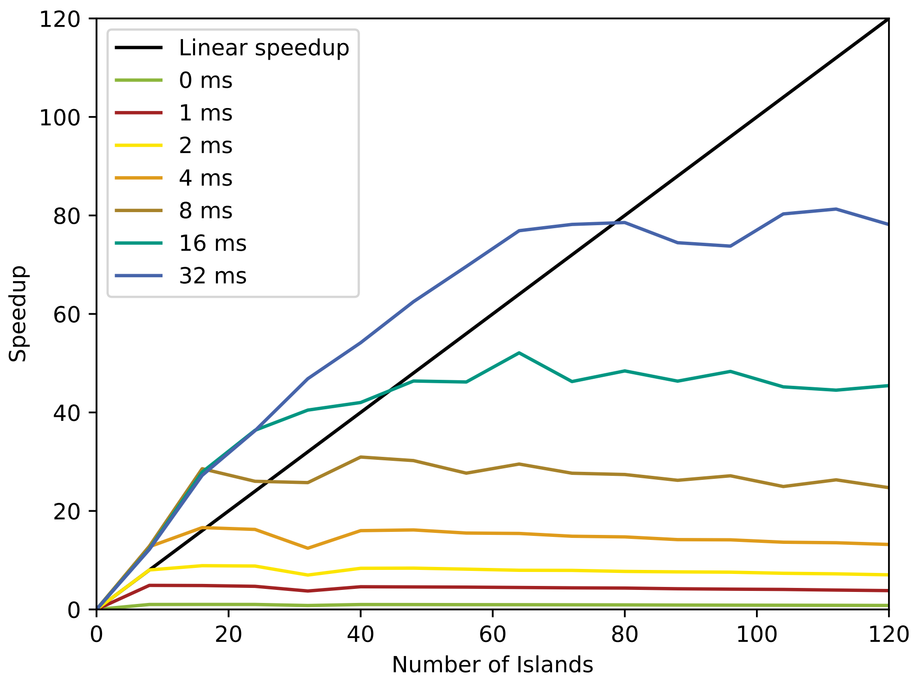


**Figure 4.28.:** Speedup with 4 migrants and a Complete topology, delay varied between 0 and 32 ms

For the obtained superlinear speedup, the implementation is the main source since the computed data structures of the subpopulations are smaller compared to the sequential EA GLEAM. Furthermore, certain genetic operators such as crossover are not executed in GLEAM, if the parents are too similar. This case usually occurs earlier with small subpopulations than with larger ones reducing the number of evaluations and therefore the computing time as shown in Table 4.23. It shows that GLEAM performs more evaluations when the population is unsplit than when it is divided into subpopulations.

The second and third reasons are not responsible for the conducted measurements since the quality of the proposed solution is not considered in this evaluation (see, 4.4)). Furthermore, the sequential and the parallel EA use the same hardware. The speedup starts to decrease at a certain point showing a constant trend. This is due to the fact that the overhead of the increased number of islands exceeds the gain in the execution time of the EA GLEAM.

| | **Ring** | **Bi Ring** | **Ladder** | **Complete** |
|---|---|---|---|---|
| **1 Island** | 4833 | 4833 | 4833 | 4833 |
| **8 Islands** | 317 | 281 | 285 | 291 |
| **16 Islands** | 137 | 142 | 145 | 147 |
| **24 Islands** | 102 | 94 | 98 | 98 |
| **32 Islands** | 76 | 75 | 70 | 72 |
| **40 Islands** | 71 | 70 | 61 | 60 |
| **48 Islands** | 58 | 54 | 47 | 48 |
| **56 Islands** | 53 | 48 | 44 | 43 |
| **64 Islands** | 54 | 43 | 40 | 34 |
| **72 Islands** | 41 | 41 | 34 | 30 |
| **80 Islands** | 41 | 39 | 31 | 29 |
| **88 Islands** | 39 | 32 | 33 | 28 |
| **96 Islands** | 45 | 35 | 26 | 24 |
| **104 Islands** | 34 | 28 | 30 | 25 |
| **112 Islands** | 36 | 25 | 26 | 22 |
| **120 Islands** | 34 | 26 | 24 | 23 |

**Table 4.23.:** Number of evaluations per island for each topology to solve the Rastrigin function with GLEAM where the stopping criteria is the number of generations which is set to 100 and the whole population consists of 1024 individuals

## 4.5. Summary

In this chapter, BeeNestOpt.IAI is presented as a new, generic and modular software solution for parallelization EAs in a cluster computing environment. Three lightweight technologies, namely microservices, container virtualization and the publish/subscribe messaging paradigm are used to develop BeeNestOpt.IAI. We started by introducing the conceptual architecture in Section 4.1. Then, we mapped the Global Model to BeeNestOpt.IAI by designing the required microservices in Section 4.2. In Section 4.3, we explained how the microservices that realize the Global Model are extended to support the Coarse-Grained Model. A modern runtime technology, namely the container virtualization is utilized to deploy each microservice into one or more containers to perform its task. In Section 4.4, we introduced our concept to evaluate the applicability and the performance of BeeNestOpt.IAI. We described the EA GLEAM used as a test case in Section 4.4.1. Thereafter, we explained the integration of GLEAM into BeeNestOpt.IAI and the successful deployment on a cluster with 4 nodes and 128 cores for benchmarking in Section 4.4.2. The communication overhead and the speedup for the Global Model is measured. The concluded results presented in Section 4.4.3 show that the underlying microservice architecture combined with container technologies introduces a linear increase of the overhead with growing population sizes and an almost linear increase of the overhead with the number of workers. Moreover, using BeeNestOpt.IAI with the Global Model and the given hardware is a good choice if the evaluation time is greater than 0.1 seconds per individual. In Sections 4.4.4.1 and 4.4.4.2, we introduced our approach to assess the performance of BeeNestOpt.IAI in terms of the migration overhead and the speedup for the Coarse-Grained Model. Several setup configurations with four topologies, namely Ring, Bi Ring, Ladder and Complete are investigated. The observed constant migration overhead for low and moderate migration rates and the high speedup for more time-consuming fitness evaluations introduce the new proposed BeeNestOpt.IAI as a promising software architecture to parallelize EAs according to the Coarse-Grained Model in a cluster environment.

# 5. Enhancing the Parallel Performance of Evolutionary Algorithms in Cluster Computing Environments

Each parallelization model of the three basic models, namely the Global Model (Master-Worker Model), the Coarse-Grained Model (Island Model) and the Fine-Grained Model has its own advantages and disadvantages. While the advantages introduce each parallelization model as a proper model for some kind of optimization problems, the disadvantages limit its common applicability. For example, the main disadvantage of the Global Model is the communication overhead which limits its application to optimization problems with high fitness computational costs and low communication demands. Otherwise, the communication overhead might outweigh the speedup gained through parallelization [68]. Another example is the inability of the Coarse-Grained Model to work efficiently with external tools and frameworks, limiting its usage to optimization problems that do not require, e.g., simulators. Obviously, the disadvantages of each parallelization model of EAs decrease the usability of the parallel EAs by limiting the parallel performance and their applicability. In order to reduce the negative effects, some characteristics of two or more of the three basic parallelization models can be hierarchically combined to form the Hierarchical Model (so-called Hybrid Model) aiming at minimizing the disadvantages and increasing the benefits [68]. Generally, the Hierarchical Model combines (i.e., hybridizes) the Coarse-Grained Model on the upper layer and one of the three basic models on the

Parts of this chapter are reproduced from:

- Khalloof, H., Mohammad, M., Shahoud, S., Duepmeier, C., & Hagenmeyer, V. A generic flexible and scalable framework for hierarchical parallelization of population-based metaheuristics. In Proceedings of the 12th International Conference on Management of Digital EcoSystems. (2020), pp. 124–131. DOI: 10.1145/3415958.3433041
- Khalloof, H., Mohammad, M., Shahoud, S., Duepmeier, C., & Hagenmeyer, V. A generic flexible and scalable framework for hierarchical parallelization of population-based metaheuristics. In Internet of Things, Vol. 16 (2021), p. 100433. DOI: 10.1016/j.iot.2021.100433
- Khalloof, H., Jakob, Shahoud, S., Duepmeier, C.,& Hagenmeyer, V . “A Generic Scalable Method for Scheduling Distributed Energy Resources Using Parallelized Population-Based Metaheuristics”. In: Proceedings of the Future Technologies Conference. (2020), pp. 1–21. DOI: 10.1007/978-3-030-63089-8_1

lower layers. E.g., combining the Coarse-Grained Model with the Global Model forms the Coarse-Grained - Global Hybrid Model. Other combinations are also possible resulting in five Hierarchical Models:

- Coarse-Grained - Coarse-Grained Hybrid Mode.
- Coarse-Grained - Fine-Grained Hybrid Model.
- Global - Fine-Grained Hybrid Model.
- Coarse-Grained - Global - Fine-Grained Hybrid Model.

In this chapter, we extend the architecture of BeeNestOpt.IAI to support the Hierarchical Model of EAs in cluster and cloud computing environments answering the research question **[RQ2]**. This introduces BeeNestOpt.IAI as a new generic and flexible software solution to combine arbitrary models from the basic parallelization models of EAs with minimum adaptation efforts. Such a combination paves the road to enhance the parallel performance of EAs by achieving further acceleration. For analyzing the practicality of the proposed solution, the Coarse-Grained - Global Hybrid Model is mapped to the microservices and container-based architecture of BeeNestOpt.IAI. By combining the Coarse-Grained Model with the Global Model, we achieve a further parallelization level by using the Global Model and maintain the advantages of the Coarse-Grained Model, i.e., the promotion of a longer preservation of genotypic diversity in the population, the associated strengthening of the breadth search and the resulting reduction of the risk of premature convergence. Moreover, the Global Model extends the application space of the Coarse-Grained Model by paving the road for using it with simulation-based optimization problems. In order to compare the Hierarchical Model with its basic parallelization models, a real-world optimization problem, namely the problem of scheduling Distributed Energy Resources (DERs) is considered as a test case.

The new approach of BeeNestOpt.IAI for hierarchical parallelization of EAs is described in Section 5.1 where in Sections 5.1.1 and 5.1.2, the mapping of the Coarse-Grained - Global Hybrid Model to BeeNestOpt.IAI and the execution workflow are introduced. In Section 5.2, the evaluation concept is introduced. The problem of scheduling DERs acting as a use case for the evaluation is presented in Section 5.2.1. In Section 5.2.2, we describe the two datasets used in the context of this evaluation. Then, we give a brief description of the application of GLEAM to the scheduling task of DERs in Section 5.2.3. We introduce the experimental configurations and results for evaluating the performance of BeeNestOpt.IAI for the Coarse-Grained - Global Hybrid Model Section 5.2.4. The research contributions presented in this chapter are the main topics of our papers [101, 102, 103].

## 5.1. BeeNestOpt.IAI for Hierarchical Parallelization of Evolutionary Algorithms

The conceptual microservice and container-based architecture of BeeNestOpt.IAI introduced in Section 4.1 enables an easy integration for the Hierarchical Model. This can be achieved by adding new microservices or adjusting the existing ones. In this Section, we explain in detail how BeeNestOpt.IAI maps the Hierarchical Model, e.g., the Coarse-Grained - Global Hybrid Model to its architecture with minimal adaptation efforts. Furthermore, the execution workflow for the Coarse-Grained - Global Hybrid Model is introduced.

### 5.1.1. Mapping the Coarse-Grained - Global Hybrid Model to the BeeNestOpt.IAI Architecture

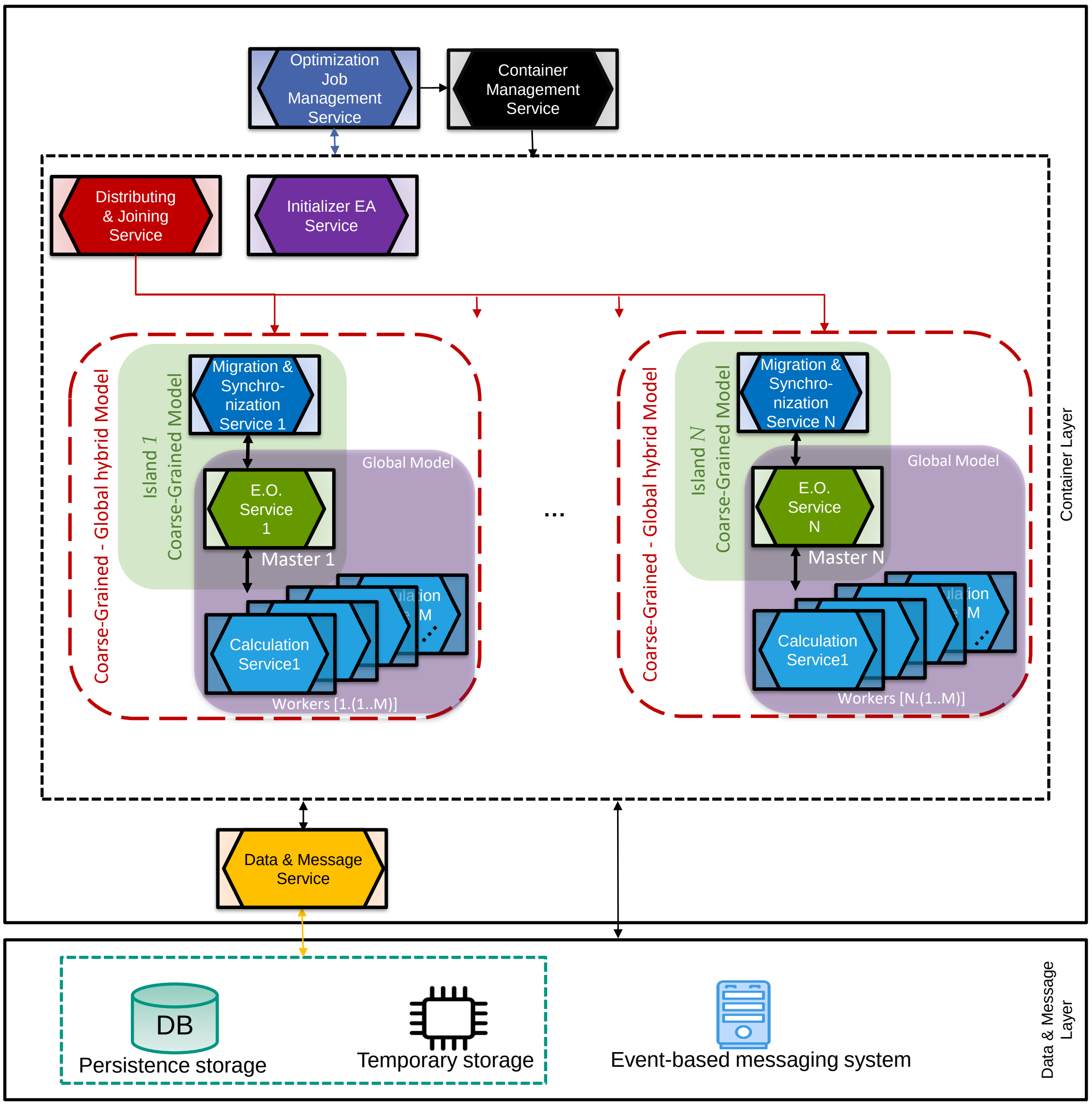


**Figure 5.1.:** Container and Data & Message Layers of BeeNestOpt.IAI for the Coarse-Grained - Global Hybrid Model with $N$ islands and $M$ workers

As explained in Chapter 4 the services for parallelizing EAs are located in the Container Layer performing all functionalities related to different EA parallelization models. Besides the basic services of BeeNestOpt.IAI introduced in Section 4.1.1, the model-related microservices described in Sections 4.2.1 and 4.3.1 are used to support the Coarse-Grained - Global Hybrid Model. These services perform, e.g., the genetic operators for the Coarse-Grained Model and the Global Model, the migration policy between the islands for the Coarse-Grained Model and apply the communication between the master and workers for the Global Model. Since the Coarse-Grained Model and the Global Model share some common functionalities and differ in others, we grouped the model-related services into three main groups.

**Shared Services** They support the shared functionalities for both EA parallelization models.

- Distributing & Joining Service (Ds.Jo. Service).
- Evolutionary Operators Service (E.O. Service).

**Coarse-Grained Model Services** They perform the tasks related to the Coarse-Grained Model.

- Migration & Synchronization Service (Mi.Sy. Service).
- Initializer EA Service (In. EA Service).

**Global Model Services** They perform the tasks related to the Global Model.

- Calculation Service (Ca. Service).

These services use REST APIs and publish/subscribe paradigm to communicate and share data among them. As shown in Figure 5.1, the E.O. Service in conjunction with the Mi.Sy. Service performs the tasks of an island in the Coarse-Grained Model. Furthermore, the E.O. Service acts as a master in the Global Model, where it communicates with the workers (Ca. Service instances) building the Coarse-Grained - Global Hybrid Model. The APIs and the publish/subscribe channels of the Mi.Sy. Service and In. EA Service described in Tables 4.14 and 4.17 are used to support the Coarse-Grained - Global Hybrid Model without any change. However, the functionalities of the shared services and the Ca. Service are modified for supporting the Coarse-Grained - Global Hybrid Model as we explain in the following. The Data & Message Layer has the same functionalities explained in Section 4.1 and 4.3 providing more flexibility and scalability as the island and worker instances are completely decoupled from each other. For simplicity's sake, the term island is used to refer to the Mi.Sy. Service and the E.O. Service.

**Distributing & Joining Service (Ds.Jo. Service)** It is extended to execute the same tasks of splitting the population, joining the results and distributing configurations for both models. Firstly, it distributes the configurations over the islands and over the workers within each island. Secondly, it splits the population into subpopulations and distributes them over the islands. Finally, it splits each subpopulation according to the number of workers within

| Method | Address | Description |
|---|---|---|
| PUBLISH | population.job.{job-id}. island.{island-id}. worker.{worker-id} | Publish subpopulation to each worker within an island. |
| PUBLISH | config.job.{job-id}. island.{island-id}. worker.{worker-id} | Publish configuration to each worker within an island. |
| PUBLISH | result.part.job.{job-id}. island.{island-id} | Publish the joined partial result of the Global Model within an island. |
| SUBSCRIBE | result.part.job.{job-id}. island.{island-id}. worker.{worker-id} | Get the partial result related to each worker within an island. |

**Table 5.1.:** Publish/subscribe channels of the Ds.Jo. Service for the Coarse-Grained - Global Hybrid Model

the island and distributes them over the workers. By joining the results, it firstly joins the partial results coming from the workers of each island to form the intermediate result. At the end of one optimization job, the final result is formed by joining the intermediate result coming from all islands to be sent to the Opt.J.M. Service.

The APIs and the publish/subscribe channels summarized in Tables 4.8 and 4.15 are used without any change. However, the publish/subscribe channels described in Table 4.9 are modified as shown in Table 5.1.

**Evolutionary Operators Service (E.O. Service)** It is a shared service that supports both models, i.e., the Coarse-Grained Model and the Global Model. While it acts as a master by applying the genetic operators, selecting the parents of the offspring and selecting of the surviving offspring to the next generation in the Global Model, it sends the intermediate population to the Mi.Sy. Service after each epoch in the Coarse-Grained Model. In Addition to the publish/subscribe channels introduced in Table 4.16, a new publish/subscribe channel is added to support the Coarse-Grained - Global Hybrid Model as described in Table 5.2.

| Method | Address | Description |
|---|---|---|
| SUBSCRIBE | result.part.job.{job-id}. island.{island-id} | Get the partial result related to the Global Model within an island. |

**Table 5.2.:** Publish/subscribe channel of the E.O. Service for the Coarse-Grained - Global Hybrid Model

**Calculation Service (Ca. Service)** The Ca. Service performs the same tasks explained in Section 4.2.1. However, its publish/subscribe channels listed in Table 4.10 are adapted to support the Coarse-Grained - Global Hybrid Model as shown in Table 5.3.

| Method | Address | Description |
|---|---|---|
| SUBSCRIBE | population.job.{job-id}. island.{island-id}. worker.{worker-id} | Get the input, i.e., the subpopulations of each worker within an island. |
| SUBSCRIBE | config.job.{job-id}. island.{island-id}. worker.{worker-id} | Get the configurations, e.g., the objective function and constraints for each worker within an island. |
| PUBLISH | result.part.job.{job-id}. island.{island-id}. worker.{worker-id} | Publish the partial result of evaluation performed by each worker within an island. |

**Table 5.3.:** The publish/subscribe channels of the Ca. Service for the Coarse-Grained - Global Hybrid Model

### 5.1.2. Execution Workflow

The execution workflow for the Coarse-Grained - Global Hybrid Model has also three phases, namely the Initialization Phase, the Iterative Evolution Phase and the Termination Phase as the ones introduced in Sections 4.2.3 and 4.3.3. Since all steps of the Coarse-Grained Model are required, only the new added ones related to the Coarse-Grained - Global Hybrid Model are explained in the following.

**Initialization Phase** It performs the same tasks as in the Coarse-Grained Model which is explained in Section 4.3.3. However, it creates not only the required containers for the island instances but also for the workers. In this phase, the container creation must be performed in a specific order for ensuring that the creation of the containers for all required services is done before starting the evolution process. First, the E.O. Service instances must be created. Then, the corresponding Mi.Sy. Service instances are booted up. Once all islands are created, the Opt.J.M. Service sends a request to the Co.Ma. Service to create the containers for the Ca. Service instances. When the components of the island, namely the E.O. Service and the Mi.Sy. Service and the corresponding Ca. Services are initialized, the Mi.Sy. Service sends an initialization signal to the Opt.J.M. Service to start the next stage.

**Iterative Evolution Phase** Before starting the Iterative Evolution Phase, the Opt.J.M. Service guarantees that all islands and workers received the related configurations that control the evolving process. As in the Coarse-Grained Model, the Opt.J.M. Service sends a start signal to all Mi.Sy. Service instances for starting an epoch. Each island sends its subpopulation to the Ds.Jo. Service which splits and distributes them over the Ca. Service instances to be evaluated. After the evaluation is completed, each Ca. Service publishes the partial results for joining them by the Ds.Jo. Service into one intermediate result for each island. After that, the Ds.Jo. Service publishes the intermediate results to the corresponding channels of the E.O. Services to apply the genetic operators. After an epoch, the migration process takes place, where each island selects and publishes its migrants and receives the

| Services | Initialization Phase | Iterative Evolution Phase | Termination Phase |
|---|---|---|---|
| Opt.J.M. Service | Yes | No | Yes |
| In. EA Service | Yes | No | No |
| Ds.Jo. Service | Yes | Yes | Yes |
| Co.Ma. Service | Yes | No | No |
| Mi.Sy. Service | Yes | Yes | No |
| E.O. Service | No | Yes | Yes |
| Ca. Service | No | Yes | No |

**Table 5.4.:** Mapping the microservices of the Coarse-Grained - Global Hybrid Model to the three execution phases

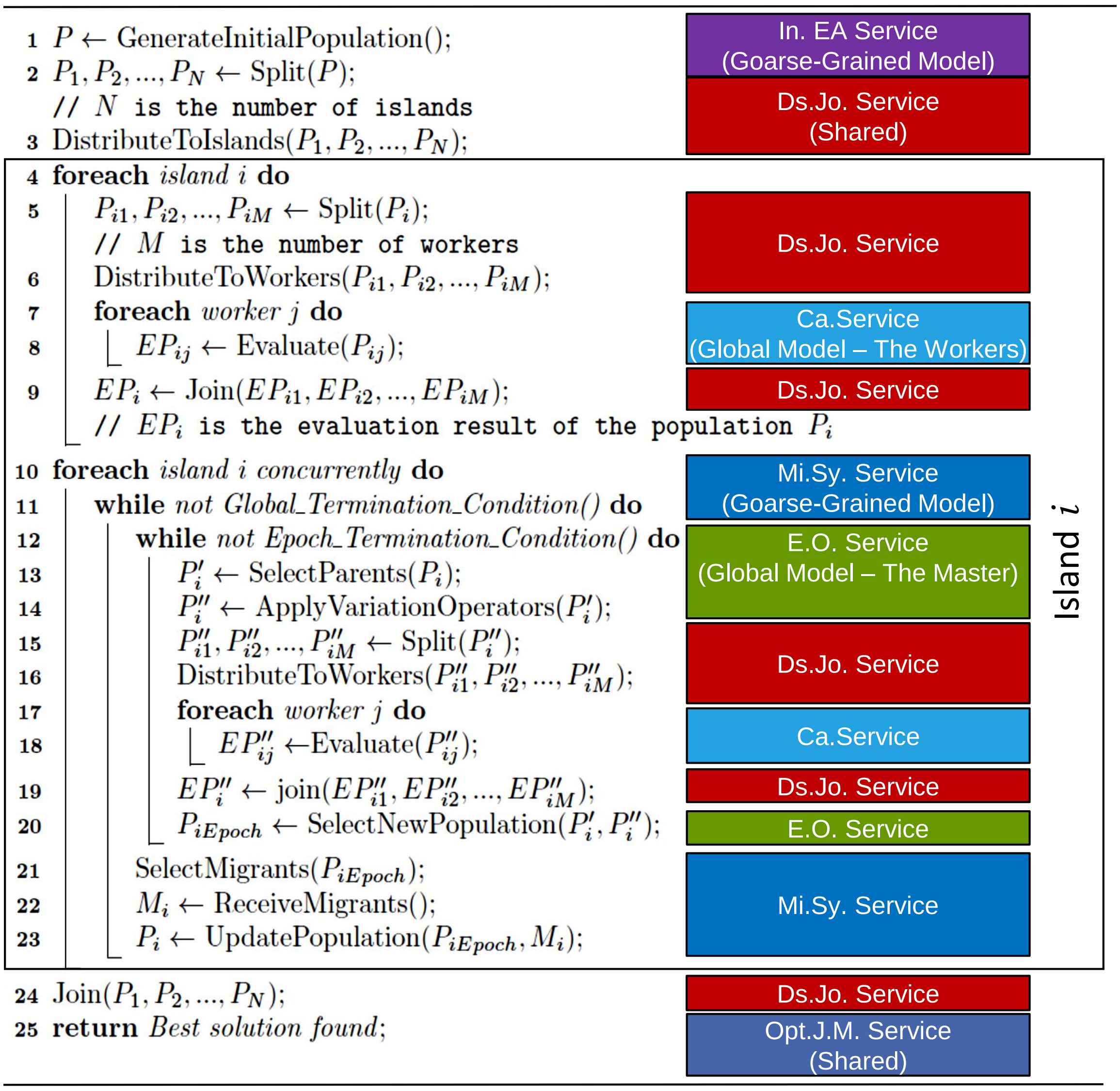
migrants from the neighbors. If the epoch termination criterion is not met, a new iteration is started by splitting the newly created subpopulation and distributing them over the islands.

**Termination Phase** The same tasks of the termination phase of the Coarse-Grained Model explained in Section 4.3.3 are performed here.

Table 5.4 summarizes how the above microservices are mapped to the three stages of the execution workflow.

Figure 5.2 shows how the basic and model-related microservices are mapped to the pseudocode of the Coarse-Grained - Global Hybrid Model. The In. EA Service creates an initial population by executing line 1. Then, the Ds.Jo. Service splits the generated initial population and distributes it to the islands by executing lines 2 and 3. After that, within each island, the Ds.Jo. Service splits the subpopulations and distributes them to the corresponding workers as in lines 4, 5 and 6 of the pseudocode. Thereafter, the Ca. Service instances start the evaluation of the subpopulations in lines 7 and 8. Once the evaluation is done, the Ds.Jo. Service joins the partial results of all workers related to each subpopulation to form the intermediate results as expressed in line 9. In lines 10 and 11, the Mi.Sy. Service checks if the global termination criterion is met or not. In the latter case, the E.O. Service selects the parents and applies the genetic operators on the subpopulation in the lines 12, 13 and 14. Lines 12, 13, 14, 15, 16, 17, 18, 19 and 20 are executed by the Ds.Jo. Service, the Ca. Service and the E.O. Service over and over until the epoch termination criterion is reached. Once an epoch termination criterion is met, the Mi.Sy. Service executes lines 21, 22 and 23 to perform the migration process by selecting the migrants and publishing them. These steps, namely from line 11 to 23 continue until a global termination criterion is met. After that, the Ds.Jo. Service joins the intermediate results of the islands as shown in line 24 and sends them to the Opt.J.M. Service which then returns the best solution for the optimization job by performing line 25.

Table 5.5 summarizes the modified publish/subscribe channels to support the Coarse-Grained - Global Hybrid Model.



```
1  $P \leftarrow$ GenerateInitialPopulation();
2  $P_1, P_2, ..., P_N \leftarrow$ Split($P$);
   // $N$ is the number of islands
3  DistributeToIslands($P_1, P_2, ..., P_N$);
4  foreach island $i$ do
5      $P_{i1}, P_{i2}, ..., P_{iM} \leftarrow$ Split($P_i$);
       // $M$ is the number of workers
6      DistributeToWorkers($P_{i1}, P_{i2}, ..., P_{iM}$);
7      foreach worker $j$ do
8          $EP_{ij} \leftarrow$ Evaluate($P_{ij}$);
9      $EP_i \leftarrow$ Join($EP_{i1}, EP_{i2}, ..., EP_{iM}$);
       // $EP_i$ is the evaluation result of the population $P_i$
10 foreach island $i$ concurrently do
11     while not Global_Termination_Condition() do
12         while not Epoch_Termination_Condition() do
13             $P'_i \leftarrow$ SelectParents($P_i$);
14             $P''_i \leftarrow$ ApplyVariationOperators($P'_i$);
15             $P''_{i1}, P''_{i2}, ..., P''_{iM} \leftarrow$ Split($P''_i$);
16             DistributeToWorkers($P''_{i1}, P''_{i2}, ..., P''_{iM}$);
17             foreach worker $j$ do
18                 $EP''_{ij} \leftarrow$Evaluate($P''_{ij}$);
19             $EP''_i \leftarrow$ join($EP''_{i1}, EP''_{i2}, ..., EP''_{iM}$);
20             $P_{iEpoch} \leftarrow$ SelectNewPopulation($P'_i, P''_i$);
21         SelectMigrants($P_{iEpoch}$);
22         $M_i \leftarrow$ ReceiveMigrants();
23         $P_i \leftarrow$ UpdatePopulation($P_{iEpoch}, M_i$);
24 Join($P_1, P_2, ..., P_N$);
25 return Best solution found;
```

In. EA Service (Goarse-Grained Model)
Ds.Jo. Service (Shared)
Ds.Jo. Service
Ca.Service (Global Model – The Workers)
Ds.Jo. Service
Mi.Sy. Service (Goarse-Grained Model)
E.O. Service (Global Model – The Master)
Ds.Jo. Service
Ca.Service
Ds.Jo. Service
E.O. Service
Mi.Sy. Service
Ds.Jo. Service
Opt.J.M. Service (Shared)
Island $i$

**Figure 5.2.:** Mapping the proposed microservices to the pseudocode of the Coarse-Grained - Global Hybrid Model

| | Microservices of the Coarse-Grained - Global Hybrid Model | | |
|---|---|---|---|
| **publish/subscribe channels** | **E.O.** | **Ds.Jo.** | **Ca.** |
| population.job.{job-id}.island.{island-id}.worker.{worker-id} | | Pub. | Sub. |
| config.job.{job-id}.island.{island-id}.worker.{worker-id} | | Pub. | Sub. |
| result.part.job.{job-id}.island.{island-id} | Sub. | Pub. | |
| result.part.job.{job-id}.island.{island-id}.worker.{worker-id} | | Sub. | Pub. |

**Table 5.5.:** Summary of the publish/subscribe channels used by the services of the Coarse-Grained - Global Hybrid Model

## 5.2. Evaluation

In this section, we evaluate the applicability of BeeNestOpt.IAI for the Coarse-Grained - Global Hybrid Model by solving a real-world optimization problem. Moreover, we investigate and discuss how the Hierarchical Model using microservice, container virtualization and the publish/subscribe messaging paradigm enhances the parallel performance of an EA by measuring the execution time and the communication overhead. For identifying the possible gain in execution time, we compare the Coarse-Grained Model and the Global Model with the Coarse-Grained - Global Hybrid Model by changing the number of islands and workers taking into account achieving a predefined target solution quality. In this evaluation, the EA GLEAM explained in Sections 2.2.3 and 4.4.1 is also used. We start in Section 5.2.1 by describing the use case scenario for scheduling Distributed Energy Resources (DERs) which is chosen as a real-world optimization problem. In this scenario, the creation of an hourly day-ahead schedule plan for a simulated microgrid based on the Ausgrid electricity dataset [153] is chosen. The preprocessing steps to generate the adequate datasets for power generation, power consumption and electricity price are discussed in Section 5.2.2. Then, a short description of the flexible coding used by GLEAM for scheduling DERs is given in Section 5.2.3. Afterwards, the experimental setup and the obtained results from the evaluation regarding the communication overhead and execution time for the Global Model, the Coarse-Grained Model and the Coarse-Grained - Global Hybrid Model are presented in Section 5.2.4.

### 5.2.1. Use Case Scenario: Scheduling Hierarchical Distributed Energy Resources

In the last decades, new types of energy generation technologies have emerged and been integrated into the existing Electric Power System (EPS) changing its architecture from a central system to a more decentralized one [11, 29]. This transition is strongly influenced by integration more Renewable Energy Resources (RERs) into the EPS. Indeed, the high penetration level of RERs comes with its own benefits, such as fulfilling the increasing energy demand, limiting the climate changes and achieving socio-economic benefits for sustainable development [157]. However, it represents new challenges for the network operators like management, coordination and uncertainties in power generation. To face such challenges, the concept of Distributed Energy Resources (DERs) represents a promising approach for facilitating the adaptation of RERs such as PVs and wind turbines into the existing EPS [41, 144]. A DER represents a small or large-scale and self-autonomous subsystem connected to an electricity network. Moreover, it balances the energy supply and demand in a specific part of a power network by providing flexible load options or storage. DER includes both generation units such as fuel cells, micro-turbines, PVs, electrical loads (demand-response), e.g., electric vehicles or flexible heating systems and energy

https://www.ausgrid.com.au/Industry/Our-Research/Data-to-share/Solar-home-electricity-data (Accessed: 2023-01-28)

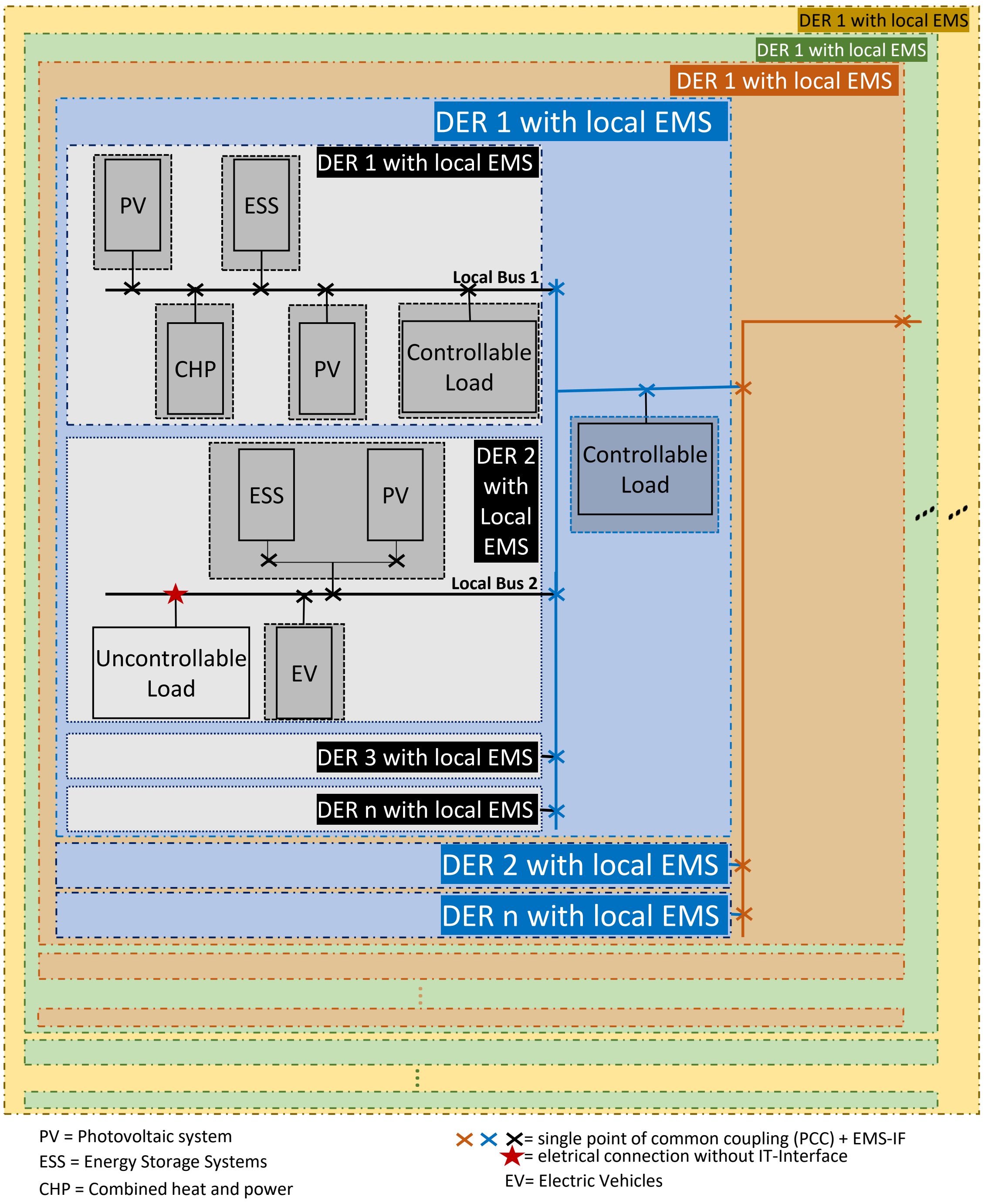


**Figure 5.3.:** Hierarchical Distributed Energy Resources (DERs) with *n* levels and local Energy Management Systems (EMS)

storage technologies like batteries. A DER may also contain uncontrollable loads which complicate the DER coordination and management. DERs interconnect bi-directionally to the grid through one or more Point(s) of Common Coupling (PCC) [74]. By the time, the high integration of DERs into the existing electrical grid will provide more clean energy generated from RERs. However, the coordination of many DERs on a large-scale level, e.g., in a distribution network or industrial area represents a challenge for the traditional Energy Management Systems (EMS). For facilitating the management process, namely the scheduling of DERs, a hierarchical approach proposed in [74, 41] is feasible. Using this approach, an energy subsystem can be structured into a hierarchical group of DERs, whereby each one can contain smaller DERs as shown in Figure 5.3. More specifically, each DER consists of components that are either elementary generation units or DERs with PCC and local EMS (see, Figure 5.3 for an example).

The blue DER1 has an EMS which manages one local load located in the same level and *n* local DERs, namely the black DER1, 2, 4, ..., *n* located in a lower level. The black DERs have EMSs that manage elementary generation units, e.g., the EMS of the DER1 manages three elementary generation units, namely two PVs and one CHP, one storage unit (ESS) and one controllable load. All elementary components and DERs have a communication interface (EMS-IF) enabling the EMSs to communicate with each other and with outside to collect data as well as for reacting to external control commands. DERs on the higher level consider the DERs located in lower ones as black boxes offering required data and an abstract service interface through their EMS-IFs.

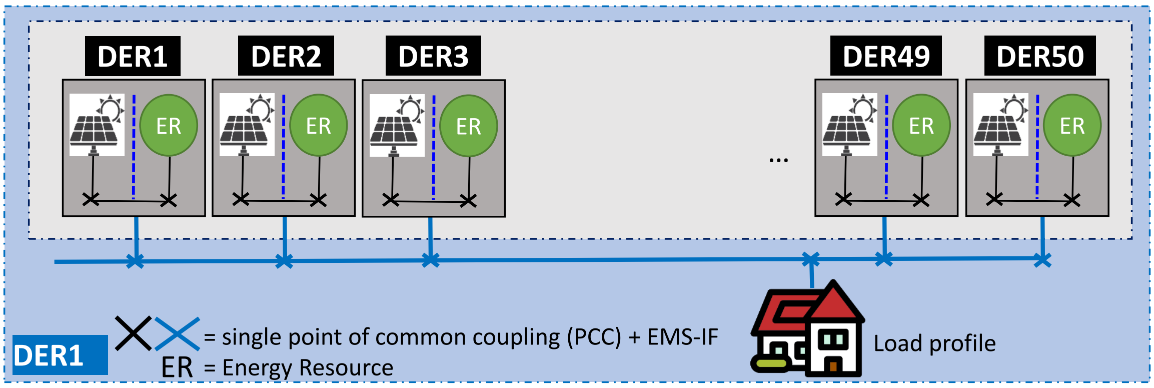


**Figure 5.4.:** Use case scenario of scheduling DERs used for evaluation

The problem of scheduling DERs according to the demands of consumers and the offers of the producers represents a real challenge for grid operators. Such a challenge lies in the uncertain nature of DERs in generating renewable energy [144]. In general, scheduling of DERs can be classified as an NP-hard optimization problem [28, 176]. Typically, scheduling of DERs problems is defined as a non-convex, large-scale, non-linear and mixed-integer optimization problem. This introduces it as a complex optimization problem to be solved by exact methods, especially if a large number of generation resources are considered [28, 176]. Consequently, the problem of scheduling DERs is an adequate test case for the proposed software architecture of parallel EAs [11]. This is due to the fact it is a complex optimization problem that cannot be solved by exact methods [77], especially if a large number of generation resources need to be considered. In the literature, there are many applications for EAs in the field of energy for, e.g., building an intelligent energy management system or solving other optimization problems such as scheduling, e.g., [16, 114, 113, 138, 137, 127, 9, 128]

The concept of hierarchical distributed energy resources is used to define a DER scheduling scenario instrumenting 50 DERs with renewable energy as shown in Figure 5.4. Through EMS-IF, each DER provides its flexibility in terms of the amount of energy that can be sold at a specific time interval with a specific price to a more or less controllable load depicted in Figure 5.4 (house symbol) located in the DER1. Each DER has two elementary generation units, namely PV and other energy resources such as batteries. For the period between 9 and 17 o'clock, the EMSs offer energy from PVs and outside this period from other resources such as batteries to cover a load profile of one customer which is depicted in Figure 5.5.

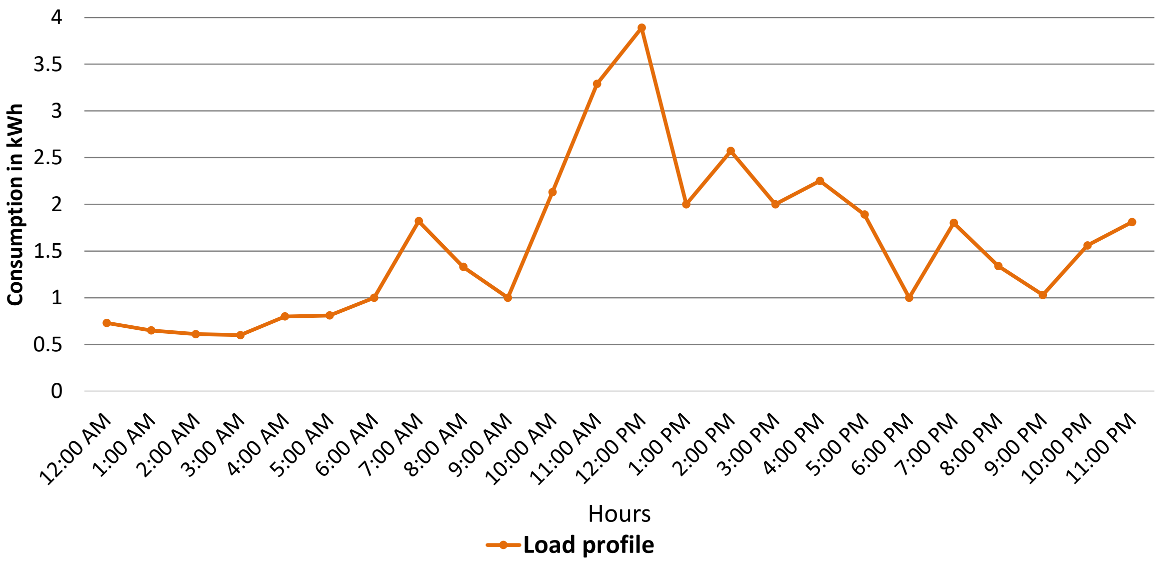


**Figure 5.5.:** The load profile used for evaluation

For defining the renewable generation behavior, the hourly real power generation data for 50 PVs provided by Ausgrid [153] is used. Figure 5.6 shows the hourly power generation aggregated over one year for the 50 DERs where each color represents one DER.

Scheduling of energy resources can be categorized into three main operation modes: Cost-Effective Operation Mode, Economic Operation Mode and Robust Operation Mode as introduced in [157]. For this evaluation, only the Cost-Effective Operation Mode is considered, aiming at minimizing the daily billing costs of the customer and covering the requested power at each time interval. A 24-hour day-ahead schedule has to be created using parallelized EAs. To this end, we define a non-linear (e.g., quadratic) cost function in Equation (5.1).

$$Cost = \sum_{i=1}^{N}\sum_{t=1}^{T} C_{i,t} * (P_{i,t}) = \sum_{i=1}^{N}\sum_{t=1}^{T} [\alpha_{i,t} * P_{i,t}^2 + \beta_{i,t} * P_{i,t} + \gamma_{i,t}] \tag{5.1}$$

Where:

- $N$ is the number of DERs.
- $T$ is the number of time intervals.

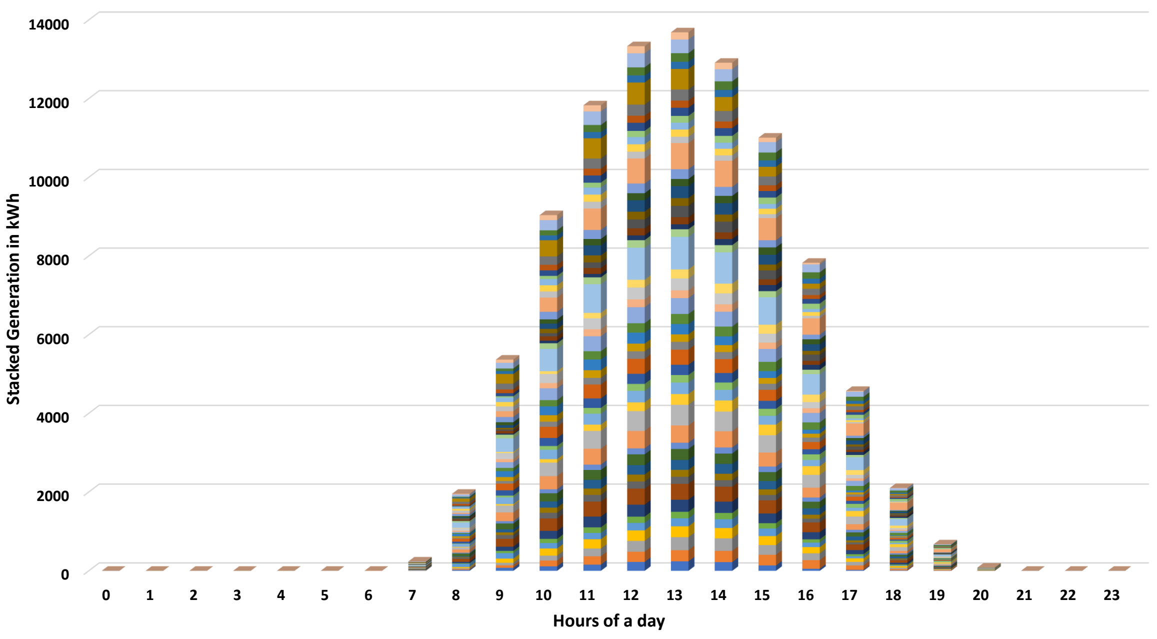


**Figure 5.6.:** 50 DERs stacked generation per hour during the first year (2010-07-01 to 2011-06-30)

- $C_{i,t}$ is the price in (EUR) for each kWh taken from resource $i$ in time interval $t$.
- $P_{i,t}$ is the scheduled power in kWh taken from resource $i$ in time interval $t$.
- $\alpha_{i,t}$, $\beta_{i,t}$ and $\gamma_{i,t}$ are the cost function coefficients defined for each DER at every time interval $t$.

Considering the necessary power balance within the blue DER of Figure 5.4 as an important factor for ensuring locally supplied power, an additional objective function, namely the Daily Total Deviation (DTD) is defined as shown in Equation (5.2). It is the sum of absolute differences between the required power and the scheduled one at every time interval $t$. For arriving at a local balance, DTD should be as low as possible.

$$DTD = \sum_{t=1}^{T} \mid \sum_{i=1}^{N} P_{i,t} - D_t \mid \tag{5.2}$$

Where:

- $D_t$ is the requested power by the laid-in time interval $t$ in kWh.

For directing the evolutionary search process to preferably find solutions without undersupply at each hour, the Hours of Undersupply (HU) function representing the number of hours of undersupply is defined as shown in Equation (5.3). It takes an integer value between zero (the optimal case no undersupply) and $T$ (the worst case where there is undersupply in all hours).

$$HU = \begin{cases} HU++, \; if \; D_t > \sum_{i=1}^{N} P_{i,t} : t \in (1, ..., T) \\ HU, otherwise \end{cases} \tag{5.3}$$

According to the above-defined functions, the optimization problem is formulated as a combinatorial non-convex mixed-integer multi-objective non-linear optimization problem. The main goal is to minimize the cost and the DTD functions as much as possible taking into account that no undersupply could happen as shown in Equations (5.4) and (5.5).

$$Minimize[Cost, DTD] \tag{5.4}$$

subject to

$$HU = 0 \tag{5.5}$$

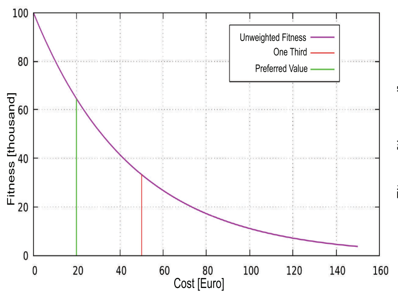
In this evaluation, the EA GLEAM is used to solve the aforementioned scheduling problem by minimizing the Cost and DTD criteria and holding the constraint HU. For calculating the fitness of each proposed solution, a weighted sum function defined in Equation (5.6) is formed to combine the results of the objective functions into a fitness value ranging between zero and 100.000 which represents the highest fitness value. The fitness value determines the probability of a chromosome to be reproduced by applying the genetic operators or discarded from the population. Precisely, the fitness value of each individual is used to choose the parents of the next generation and to decide if the offspring have to be accepted or rejected. Note that the Fitness function correlates both optimization functions Cost and DTD for directing the evolution towards a ‘certain optimization direction’ which emphasizes the DTD value in favor of the Cost value. For handling the equality constraint HU, the Penalty Function (PF) shown in Equation (5.7) which yields a value between zero and one (no undersupply) is defined. The PF penalizes solutions which tend to evolve towards a high value of under-production hours so that an undersupply of 5 hours already reduces the fitness value to a third as shown in Figure 5.7c. For this purpose, the numerical value provided by the Ca. Service instances for each criterion must be mapped to a uniform fitness scale using one of the standard normalization functions of GLEAM, namely linear, exponential and mixed linear–exponential. For the Cost and DTD criteria, the inversely proportional exponential function is used as shown in Figures 5.7a and 5.7b, respectively.

$$Fitness = (0.4 * Cost + 0.6 * DTD) * PF(HU) \tag{5.6}$$

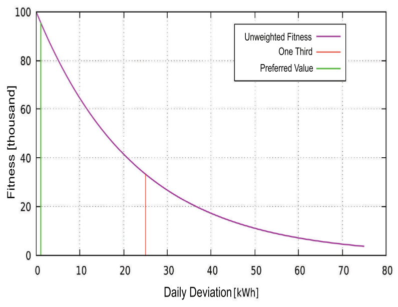
$$PF(HU) = (1 - \frac{1}{T}HU) \tag{5.7}$$

## 5.2.2. Datasets for Power Generation/Consumption and Price

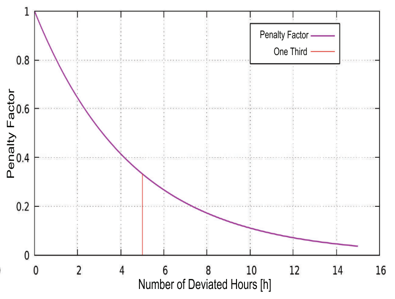
For this evaluation, we mainly use the solar home electricity dataset provided by Ausgrid [153]. Ausgrid published the energy consumption and PV production data for 300 households for the period from 1 July 2010 to 30 June 2013. For privacy purposes, all information which could lead to the identification of the customers is removed from the dataset. However, the postcode for each customer number is left. Figure 5.8 outlines the corresponding regions to the given postcodes of all residential customers.



**(a)** Cost criterion



**(b)** Daily Total Deviation (DTD) criterion



**(c)** Penalty function for the Hours of Undersupply (HU) criterion

**Figure 5.7.:** Mapping the defined objective functions to the fitness function

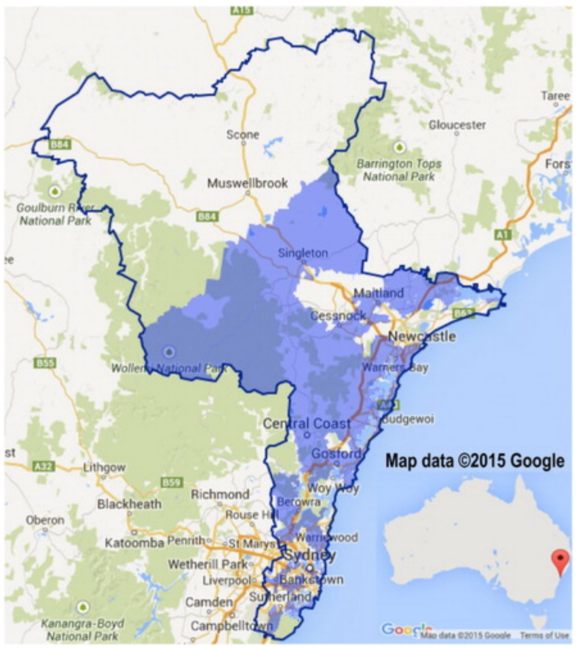


**Figure 5.8.:** Outline of the Ausgrid distribution network that covers 22,275 km and includes load centers in Sydney and regional New South Wales. Shaded regions within the Ausgrid network correspond to postcode areas included in the dataset of 300 customers [153]. Map data: 2015 Google

An overview of the dataset structure is presented in Table 5.6. The customer ID column indicates the customer identifier which ranges between 1 and 300. The generator capacity column identifies the maximum capacity of the PV installation in kilowatt-peak (kWP). The postcode column refers to the location of each customer. The next column is the consumption category which takes a value of three different two-letter codes, namely GC (General Consumption), CL (Controlled Load) and GG (Gross Generation for electricity generated by a PV). The date column has the following date format: ‘DD/MM/YYYY’. The consumption and generation data are collected in a 30-minutes interval.

For further analysis, the data is divided into two datasets, one dataset for the generated energy from the PV systems where each household serves as a potential generation resource

| Customer | Generator Capacity kWp | Postcode | Consumption Category | Date | 00:30 kWh | 01:00 kWh | ... | 23:00 kWh | 23:30 kWh |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 3.78 | 2076 | CL | 01/07/2010 | 1.25 | 1.25 | ... | 0 | 0 |
| 1 | 3.78 | 2076 | GC | 01/07/2010 | 1.077 | 669 | ... | 118 | 219 |
| 1 | 3.78 | 2076 | GG | 01/07/2010 | 0 | 0 | ... | 0 | 0 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 246 | 1.53 | 2261 | CL | 30/06/2013 | 1.465 | 0 | ... | 0 | 22 |
| 246 | 1.53 | 2261 | GC | 30/06/2013 | 1.057 | 671 | ... | 815 | 878 |
| 246 | 1.53 | 2261 | GG | 30/06/2013 | 0 | 0 | ... | 0 | 0 |

**Table 5.6.:** Format of the solar household dataset from Ausgrid

and the other one for the consumption data. The datasets are checked for missing values and completely missing entries for whole days. Over the three years, each household in each dataset should have 1096 days, i.e., 52560 data points. Some households have a lot of missing days for more than one month. Overall there are 73 households which are dropped from the datasets, since they have up to 519 missing days. After removing the respective 73 households from the datasets, 164 households with 1096 days are left for the considered use case scenarios.

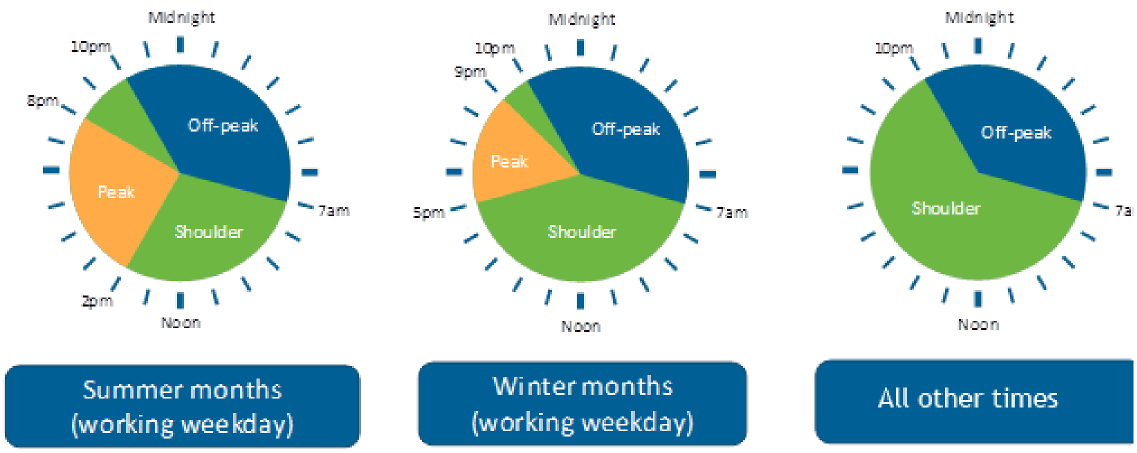


**Figure 5.9.:** Price schema for the residential customer taken from the Ausgrid website

Another required data for scheduling DERs is the price for the bought energy at each time interval. The Ausgrid dataset does not specify any price for the energy generated by the PVs of the households. In order to define a feasible price schema, we consider two resources, namely the Energy Australia website  and the Ausgrid website. These websites

https://www.ausgrid.com.au/Your-energy-use/Meters/Time-of-use-pricing (Accessed: 10.02.2023)
https://www.energyaustralia.com.au (Accessed: 10.02.2023)

offer information on the prices charged at the different parts of the day as shown in Figure 5.9. The shoulder period is applied for each day between 7:00 and 22:00 o'clock. During the shoulder period, the peak period is applied. For the rest of the day, i.e., between 22 and 7:00, the off-peak period is considered. The peak period, in which more charges are added to each consumed kilowatt-hour (kWh), is mainly located in summer and winter. According to these periods and based on the price scheme taken from both websites, we define the following three base groups for our price scheme:

- Shoulder Period: 31.02 cents per kWh.
- Peak period: 59.29 cents per kWh.
- Off-Peak Period: 18.92 cents per kWh.

Applying these prices for all resources means that every resource sells its generated energy for the same price at the same time as the others. Therefore and for a more realistic pricing scheme, upper and lower bounds for the three price groups are defined. The prices for each resource can be randomly varied within a range of 25%, 35% and 50% of the base prices for the peaks period, the shoulder period and the off-peak period, respectively as shown in Figure 5.10, 5.11 and 5.12. Consequently, obvious differences in prices between the considered resources can be observed, adding more challenges to the scheduling problem.

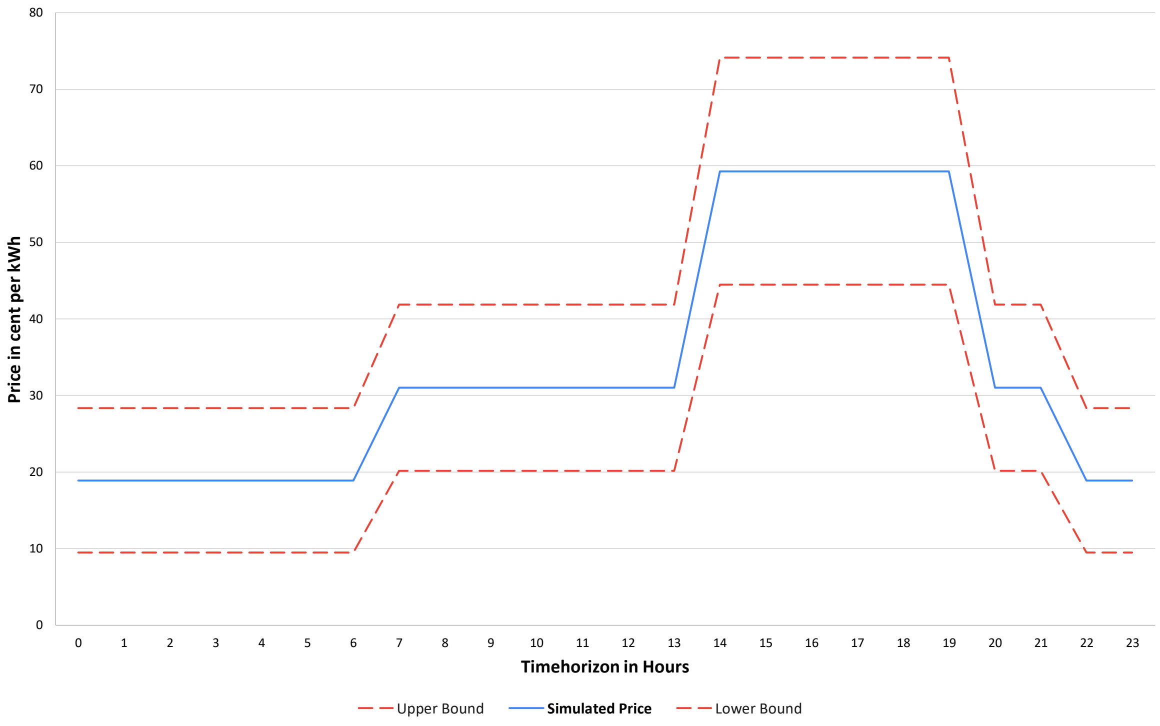


**Figure 5.10.:** Added aberration borders for pricing summer days

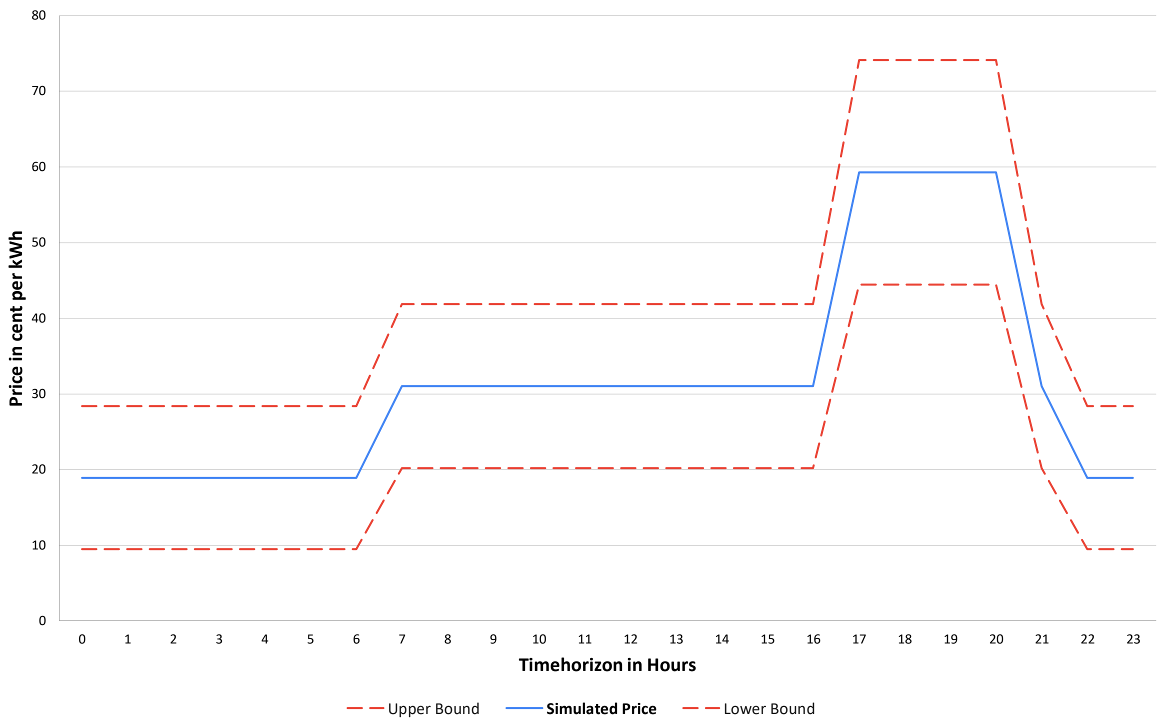


**Figure 5.11.:** Added aberration borders for pricing winter days

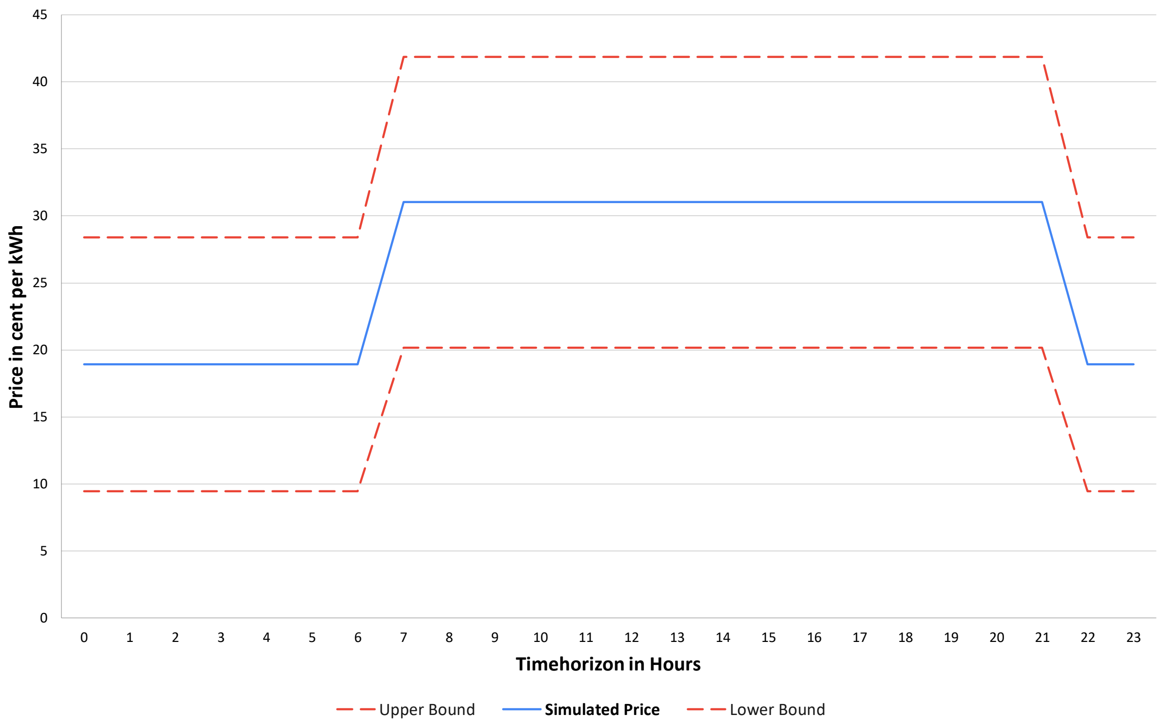
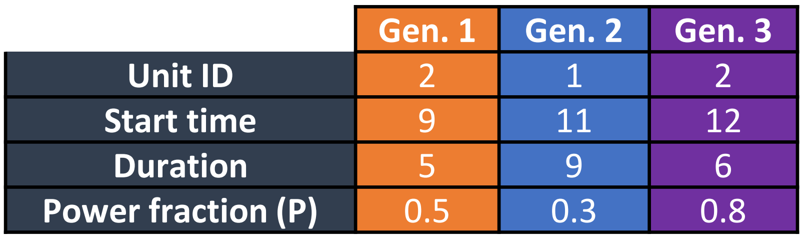


**Figure 5.12.:** Added aberration borders for pricing for other days which are not covered by the pricing during summer or winter period, i.e., 5.10 and 5.11

### 5.2.3. EA GLEAM for Scheduling DERS

We choose the EA GLEAM as one of the state-of-the-art EAs to solve general scheduling problems in several different applications, e.g., [24, 88, 89]. The flexible and generic coding schema used in GLEAM to aggregate semantically related decision variables in one gene distinguishes it from other EAs as explained in Section 2.2.3. The third type of chromosome in GLEAM which consists of a dynamic number of genes is considered for this evaluation. In this type, more than one gene of the same type may belong to a chromosome. The sequence of the genes and the length of chromosomes are subject to evolutionary change. For scheduling DERs, a dynamic number of scheduling operations is required, since it is not known a priori how many peaks and thus how many changes a demand curve has, for which the corresponding power generation is to be provided by a schedule (cf. [24, 90] for a detailed discussion). To this end, GLEAM maps each scheduling operation to a gene consisting of a gene type ID which is referred to Unit ID and the three decision variables which are depicted in Figure 5.13 (see, 2.2.3 for more details).



| | Gen. 1 | Gen. 2 | Gen. 3 |
|---|---|---|---|
| Unit ID | 2 | 1 | 2 |
| Start time | 9 | 11 | 12 |
| Duration | 5 | 9 | 6 |
| Power fraction (P) | 0.5 | 0.3 | 0.8 |

**Figure 5.13.:** Exemplary chromosome with three genes encoding a possible solution to schedule two resources

- **Unit ID** corresponds to the DER ID indicating the utilized DER. In this evaluation, the DER ID can take a value between one or fifty, since only 50 DERs are considered.

$$UnitID \in [1, 50] \ with \ UnitID \in Z \tag{5.8}$$

- **Start time** determines the start time of taking energy from this DER for a specific number of the time intervals defined by the duration variable. In this evaluation, an hourly resolution is considered for day-ahead scheduling. Therefore, it can take a value between one and 24.

$$1 \leq StartTime \leq 24 \ with \ Duration \in Z \tag{5.9}$$

- **Duration** defines the number of time intervals to which this setting is applied. It ranges also between one and 24.

$$1 \leq Duration \leq 24 \ with \ Duration \in Z \tag{5.10}$$

- **Power fraction** is a relative value – e.g. in percent of the maximum provided power – and refers to the amount of energy that is taken from the DER within the defined time intervals. The range of the variable is defined as follows.

$$1 \leq P \leq 100 \ with \ P \in R \tag{5.11}$$

An allocation matrix with *m* rows and *n* columns is created to interpret each chromosome. While the rows correspond to the number of resources, i.e., DERs, the columns refer to the number of the time intervals. Since each chromosome can contain genes that refer to the same DER, we build the allocation matrix by considering the order of them within a chromosome. Precisely, the later gene overwrites matrix entries of the previous ones with the same DER ID. After creating the allocation matrix, the relative values of the power fraction of each considered DER are converted to absolute ones by multiplying them by the corresponding power generation values, e.g., the forecasting data of renewable energy resources. As shown in Figure 5.14, the example chromosome from Figure 5.13 consisting of three genes representing an hourly day-ahead schedule is interpreted. An allocation matrix with two rows is built, since the first and the third genes have the same DER ID. In other words, the third gene overwrites the first one at the time intervals 12 and 13. For evaluation (simulation) purposes, the relative values of power fraction are transformed to absolute values by multiplying them by the corresponding values of the actual maximum power generation provided by the EMS of each DER for the corresponding time intervals.

For achieving this interpretation, an additional microservice, namely the Interpretation Service is added to the above-mentioned microservices as shown in Figure 5.15. The Interpretation Service performs all aforementioned tasks related to chromosome interpretation to generate the allocation matrix. For scheduling DERs, two types of data, namely dynamic and static data about the DER components are required. While dynamic data refers to data that is continuously changed according to different factors such as the weather, static data represents data that is fixed like the number and type of DER components, technical constraints for the conventional energy resources, e.g., minimum and maximum capacity, ramping limits and minimum up and down times. The actual state of batteries, forecasting data for the generation of RERs and consumption and market prices are examples of dynamic data. This data is stored in the Data & Message Layer, i.e., in the persistent database either automatically, if the DER has an Energy Management System Interface (EMS-IF) or manually if it does not. The Da.Me. Service plays an essential role by providing other services with both types of data. While the In. EA Service and E.O. Service need it for creating the initial population and offspring, the Interpretation Service uses it to convert the relative power values to absolute ones. The Ca. Service also needs such data to calculate the objective functions and constraints. Obviously, the interpretation process can require much computing time according to the size of population. Therefore, BeeNestOpt.IAI can deploy as many Interpretation Service instances as required, allowing a parallel interpretation for scalability.

GLEAM is deployed into two services, namely the In. EA Service and the E.O. Service. While it creates the initial population in the In. EA Service, it evolves the offspring by applying the genetic operators in the E.O. Service acting as a master of the Global Model.

### 5.2.4. Experimental Setup and Results

The applicability of BeeNestOpt.IAI by realizing the Coarse-Grained - Global Hybrid Model using microservices and container technologies to solve a real-world optimization

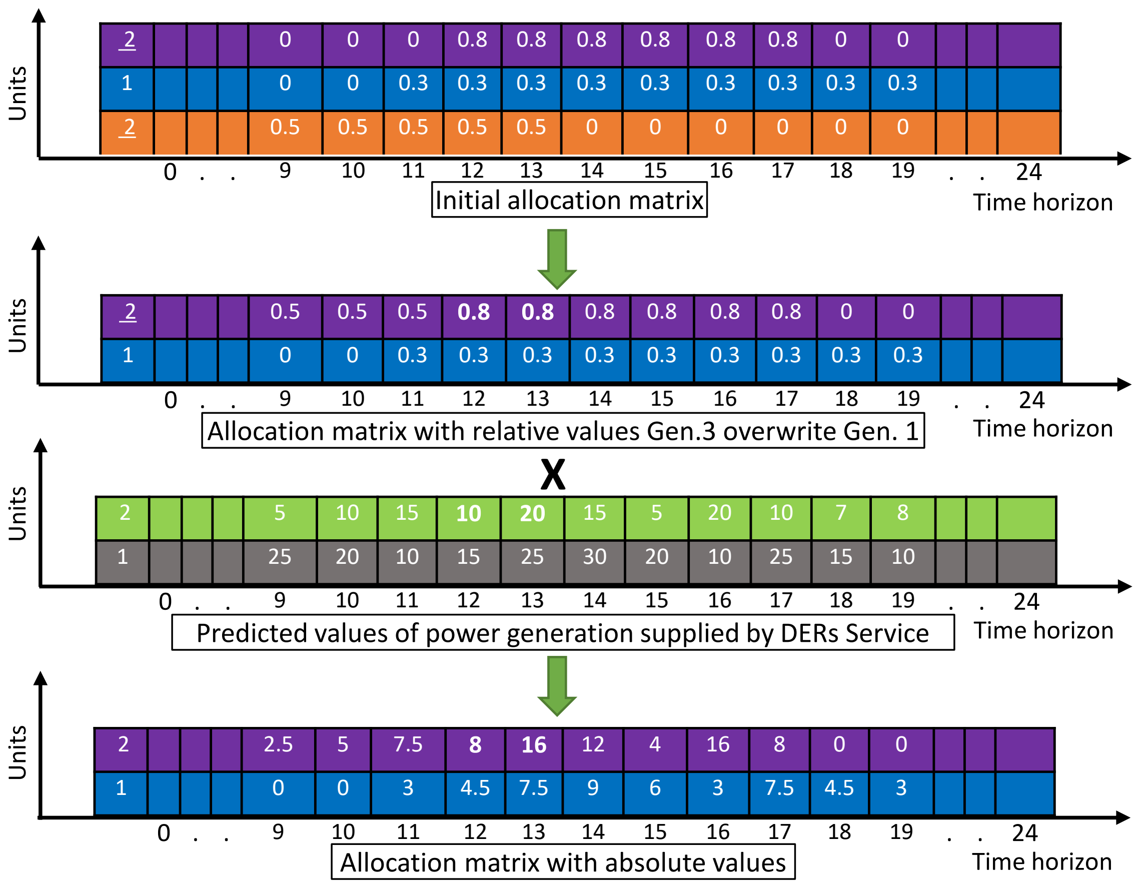

Units
Initial allocation matrix
Time horizon
Allocation matrix with relative values Gen.3 overwrite Gen. 1
X
Predicted values of power generation supplied by DERs Service
Allocation matrix with absolute values
0 . . 9 10 11 12 13 14 15 16 17 18 19 . . 24

**Figure 5.14.:** Interpretation of one chromosome with three genes for scheduling 2 DERs on a 24-hour time horizon

problem is evaluated. Precisely, we discuss the parallel performance of BeeNestOpt.IAI by investigating the execution time and the communication overhead considering the use case scenario for scheduling DERs introduced in Section 5.2.1. To achieve that, we deploy different combinations of islands and workers on the same cluster that is used for the evaluation introduced in Section 4.4.2. However, only a part of the CPUs and RAM is available for this evaluation, i.e., 68 Intel cores (2.4 GHz) and 196 GB RAM. The number of used cores is 60, since at minimum two cores are left on each node for the OS.

We assign 12.5% of the maximum power of each core for the containers running the Opt.J.M. Service, Ds.Jo. Service, Co.Ma. Service, In. EA Service, Mi.Sy. Service, E.O. Service and Da.Me. Service and 25% of the maximum power of each core for each container running the Interpretation Service and the Ca. Service. Hence, the maximum number of deployed containers cannot exceed 240 containers. For example, if we deploy one instance from each service except the Interpretation Service and Ca. Service, nine containers are required, i.e., one core and a quarter of a core are used. The rest of the cores, namely 58 ones and three quarters of a core, allow the deployment of about 116 works resulting in 232 running containers. While the number of islands and workers is varied, we keep all other EA configuration parameters fixed, aiming at achieving a fair evaluation. We set the global

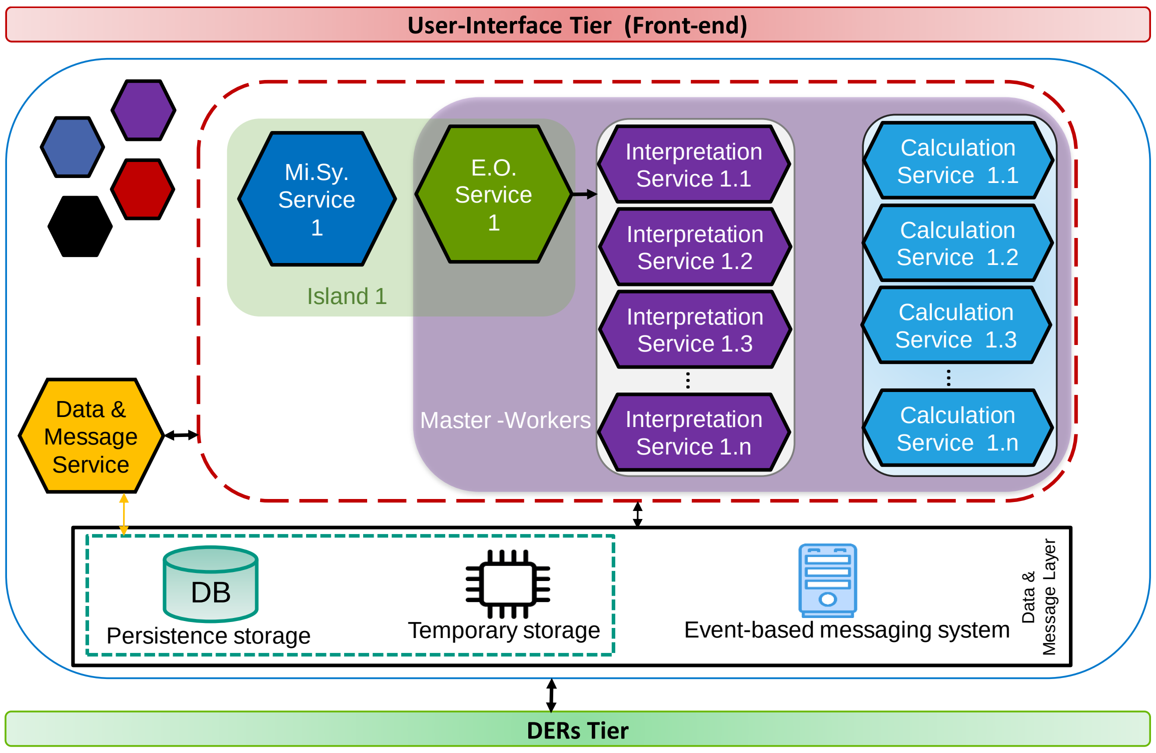


**Figure 5.15.:** The DERs Service and the Interpretation Service instances inside one island with one Master and $N$ workers

stopping criterion to solution-stop where a predefined target solution quality is given. Since it is difficult to reach the highest fitness value, namely '100.000' in a timely manner, we set the desired target solution quality to '70.000'. In other words, we consider that GLEAM has found a sufficiently good solution if it obtains 70% of the highest fitness value. This target solution quality does not represent the quality of an optimal solution rather it represents the average of good solutions that have been reached with a predefined EA effort. After testing different population sizes, a global population of 512 chromosomes is chosen to explore the search space and reach such a fitness value.

If at least one island reaches the given solution quality, the evolution process will be stopped. In the context of this evaluation, the overhead refers to the time needed for the communication among the services and by the services of BeeNestOpt.IAI themselves to perform one optimization task excluding the EA evolution time and evaluation time, i.e., the execution time of E.O. Service and Ca. Service. By comparing the Coarse-Grained - Global Hybrid Model to its both basic models, namely the Coarse-Grained Model and the Global Model, we aim at identifying a possible gain in the speedup by using the Coarse-Grained - Global Hybrid Model while keeping an acceptable communication overhead. The parallel performance of the Coarse-Grained Model and the Coarse-Grained - Global Hybrid Model is measured according to four communication topologies, namely uni-directional ring (Ring), bi-directional ring (Bi-Ring), ladder (Ladder) and a completely connected graph (Complete) and a migration rate of 10% of the subpopulation (deme) size. While the best migration selection policy is applied for choosing the migrants from each deme to be exchanged among its neighbors, the worst policy as a replacement policy is applied, enabling the replacement of the worst individuals in the target demes.

In this evaluation, we follow the following steps: Firstly, homogeneous EA configuration and a synchronous migration policy are applied. Then, heterogeneous EA configuration and an asynchronous migration policy are applied by using the combination of islands-workers that introduces the fastest execution time aiming at achieving more acceleration (see, Section 2.2.1 for more details about island homogeneity). The migration frequency (also called epochs) is executed after running 50 generations in each island. Each experiment is performed ten times aiming at limiting the effect of single delays caused by the OS or other running services and due to the non-deterministic nature of EAs. For calculating the gained speedup for the Coarse-Grained - Global Hybrid Model and its two basic models, the task of scheduling DERs is sequentially performed by deploying the EA GLEAM using only one island and one worker. This allows us to compare the execution time and the overhead of each parallelization model introduced by the proposed software solution to the sequential one. The overall execution time to reach the predefined fitness value, namely 70000 is 300522 seconds while the overhead is about 39 seconds and the container creation time is 82 seconds. The overhead and the container creation time are fairly low as compared to the execution time. These values represent the comparative values used in the experiments described below.

**Results - Global Model** In this experiment, we evaluate the Global Model by varying the number of workers between 5, 10, 15 and 20 workers. As shown in Figure 5.16, the obtained results show that the execution time decreases by deploying more workers. Compared to the sequential execution, a gain in the execution time ranging between 260377 seconds by using 5 workers and 281530 by using 20 workers is obtained. It is noticeable from Figure 5.16 that the Global Model cannot guarantee a solid increase in the gain of the execution time by deploying more workers. For example, the gain in the execution time by increasing the number of workers from 10 to 20 is smaller than the one obtained by increasing the number of workers from 5 to 10. This is due to the fact that having a master in a distributed system with too many workers leads to a well-known communication problem, namely the bottleneck problem. Indeed, the master cannot quickly react to the requests sent by the workers when most of them communicate with it at the same time.

From Figure 5.16, it is clear that the framework overhead does not increase by deploying more workers but it remains nearly stable. On the contrary, the container creation time is increased from 122 seconds by using 5 workers to 146 by using 20 workers, since more containers are created and initialized.

**Results - Coarse-Grained Model** We evaluate the Coarse-Grained Model by varying the number of islands between 5, 10, 15 and 20 islands. Four considered topologies, namely Ring, Bi-Ring, Ladder and Complete with homogeneous EA configuration and a synchronous migration are considered. Generally, the Coarse-Grained Model accelerates

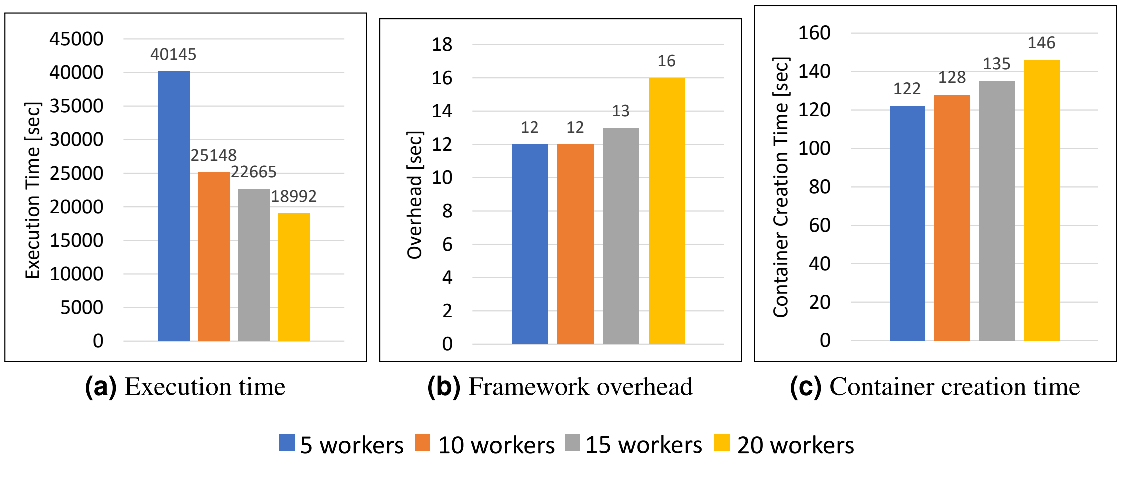


**Figure 5.16.:** Execution time, framework overhead and container creation time of the Global Model

the evolution process of GLEAM by deploying more islands and according to the communication topology. This is noticeable in Figure 5.17 which shows that the execution time is decreased for the four communication topologies by increasing the number of islands. By averaging the execution times of using 5, 10, 15 and 20 islands over the four communication topologies, it is noticeable that the averaged execution time is decreased from 80454 seconds by using 5 islands to 16757.75 seconds by using 20 islands as shown in Figure 5.18. Compared to the sequential execution, a gain of 283765 seconds in the execution time is achieved by using 20 islands. The reason behind this result lies in the fact that the size of the data structure, i.e., the number of chromosomes within each subpopulation, becomes smaller, accelerating the evolution process of each island. While Ring and Bi-Ring promote breadth-first search which is useful for finding optimal solutions consuming more time, Ladder and Complete foster depth-first search which leads faster to find good solutions but not necessarily the optimal ones. Hence, Ladder and Complete communication topologies are able to accelerate the evolution process more than Ring and Bi-Ring in most cases as shown in Figure 5.17.

By deploying more islands, more communications between the services are established in the Coarse-Grained Model for population distribution and results collection. However, a slight increase in the overhead by increasing the number of islands is observed as shown in Figure 5.19. Furthermore and as shown in Figure 5.20, the container creation time slightly increases by deploying more islands. On average, the container creation time in the Coarse-Grained Model is greater than the one of the Global Model, since more containers are required to deploy the islands. Indeed, there is no significant difference between the communication topologies, since the number of containers to be created is independent of the deployed topology.

The Coarse-Grained Model introduces relatively small overhead and container creation time compared to the execution time. The results are obtained based on a real-world optimization problem for the Global Model and the Coarse-Grained model confirm the ones presented in

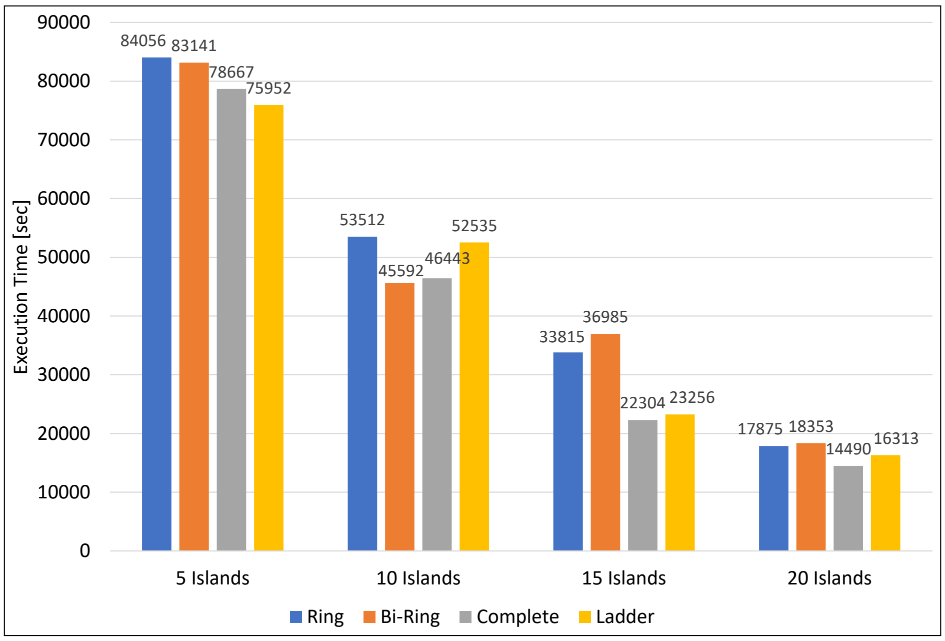


**Figure 5.17.:** Execution time of the Coarse-Grained Model

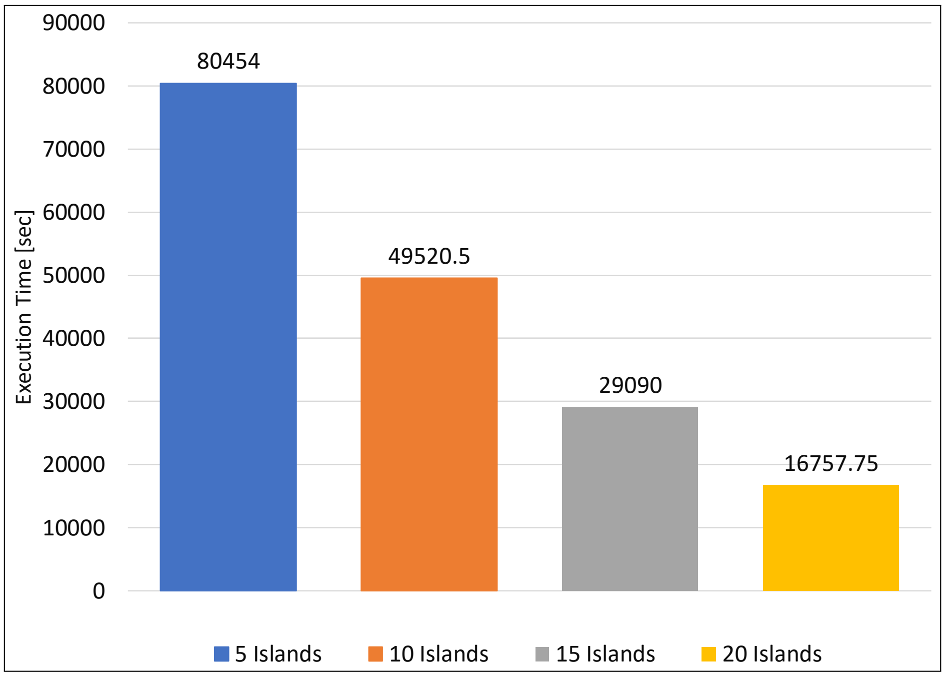


**Figure 5.18.:** Execution time of the Coarse-Grained Model averaged over the four communication topologies

Section 4.4.3 and Section 4.4.4, in which the amount of time for the fitness evaluation is artificially set.

**Results - Coarse-Grained - Global Hybrid Model** The key focus of the experiment is to assess the applicability of the proposed microservice-based solution to distribute the

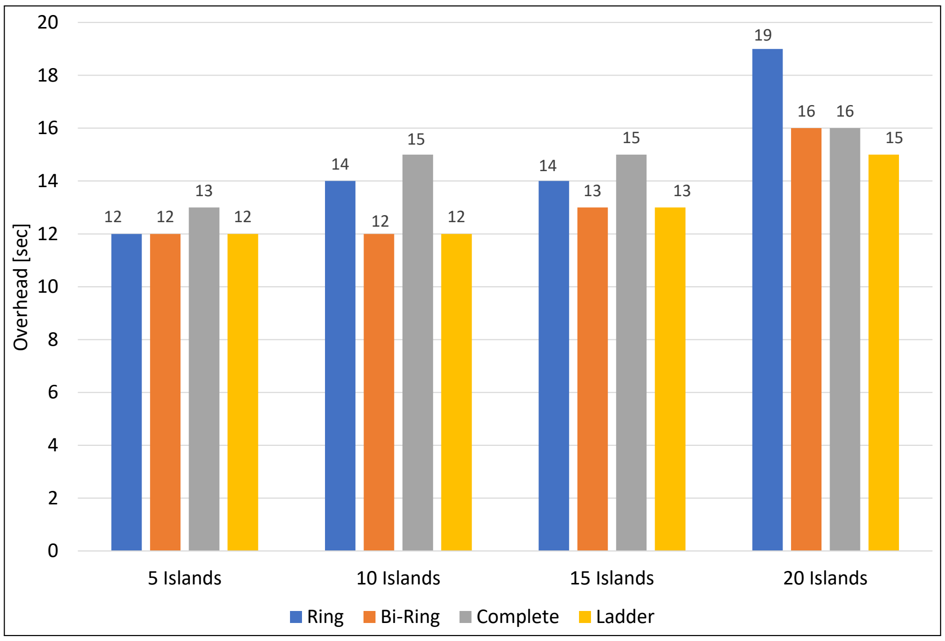


**Figure 5.19.:** Framework overhead of the Coarse-Grained Model

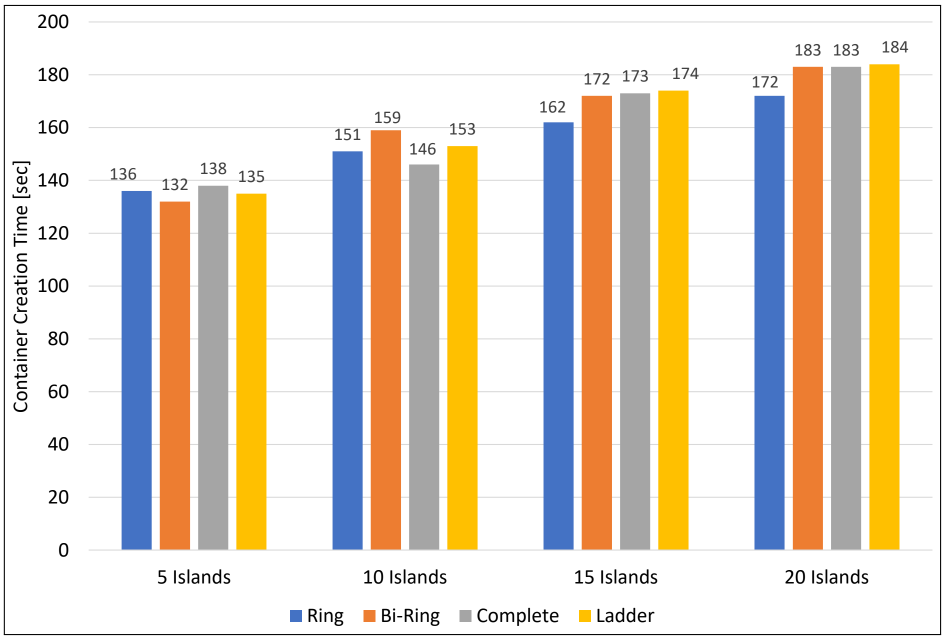


**Figure 5.20.:** Container creation time of the Coarse-Grained Model

execution of EAs according to the Hierarchical Model. To this end, the parallel performance of the Coarse-Grained - Global Hybrid Model in terms of the execution time and framework overhead is evaluated in a cluster environment. The same four communication topologies, namely Ring, Bi-Ring, Ladder and Complete used in the evaluation of the Coarse-Grained Model are also used in this experiment. For each communication topology, the number of service instances running islands and workers is varied taking into account that the whole number of containers running these microservices must not exceed the number of available computational resources. Precisely, four islands-workers combinations are defined, namely 5 islands with 20 workers per island, 10 islands with 10 workers per island, 15 islands with 7 workers per island and 20 islands with 5 workers per island. This results in 200 running containers for the first, second and fourth combinations and 240 running containers for the third one, namely 15 islands with 7 workers per island. By comparing the execution time of the islands-workers combinations for the same communication topology, we find that different distributions of available computational resources among the number of deployed islands and workers have only a limited effect on the execution time as shown in Figure 5.21.

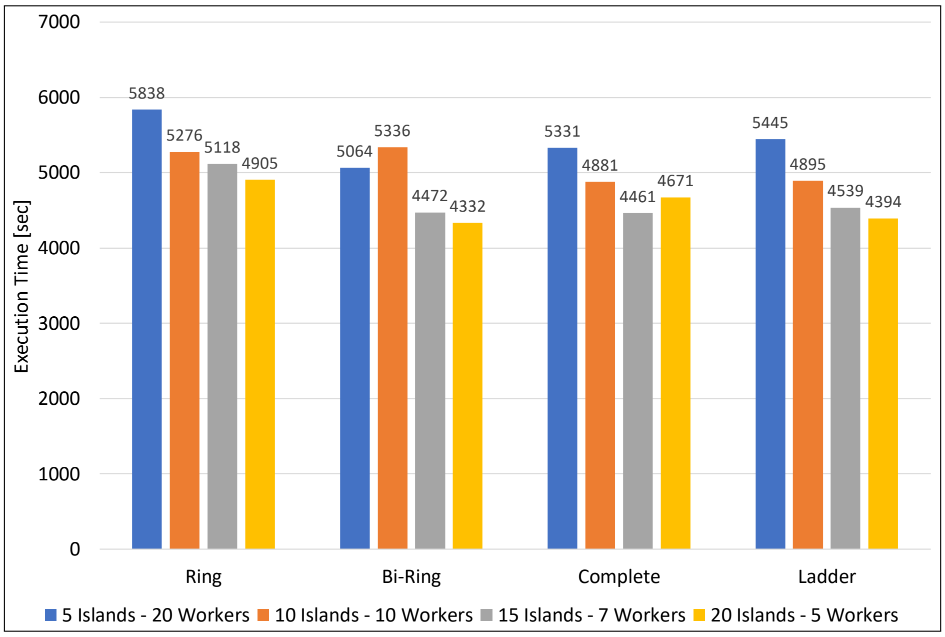


**Figure 5.21.:** Execution time of the Coarse-Grained - Global Hybrid Model

By averaging the execution times for each islands-workers combination over the four communication topologies as shown in Figure 5.22, we can conclude that deploying more islands with fewer workers decreases the execution time. For example, by using 20 islands with 5 workers, a gain in execution time of 15% and 10% can be achieved compared to 5 islands with 20 workers and 10 islands with 10 workers, respectively. Such a result can be interpreted as follows: by deploying more islands, the number of evaluations performed by GLEAM is decreased. The reason behind this decrease lies in the fact that the subpopulations and the computed data structures (number of chromosomes) are smaller compared to the global population [2]. Consequently, it is clear that the Coarse-Grained Model has a more significant positive effect on the EA execution time than the Global Model.

In order to investigate the effect of the communication topologies on the execution time of the EA, we average the execution time of each topology over the above-mentioned islands-workers combinations as shown in Figure 5.23. Obviously, the Ring topology takes more time on average than the others to solve the problem of scheduling DERs. In other words, the Bi-Ring, Ladder and Complete topologies find a solution with the same solution quality reached by the Ring topology with less time, achieving a gain in the execution time by

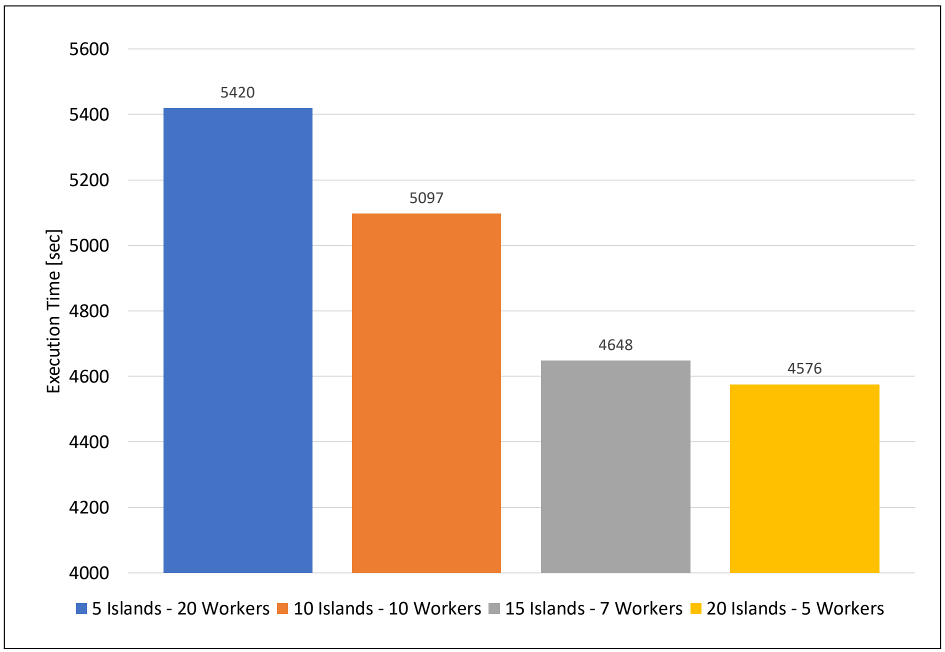


**Figure 5.22.:** Execution time of the Coarse-Grained - Global Hybrid Model for each islands-workers combination averaged over the four applied communication topologies

about 9%. This gain is obtained, since each island in the Bi-Ring, Ladder and Complete topologies communicates with two islands at minimum, enabling the possibility to spread solutions with good genetic information over the topology more rapidly. However, this is not the case for the Ring topology where each island exchanges solutions (migrants) with only one island.

Compared to the execution time, it is clear that the overhead is still acceptable and low. On average, the four communication topologies achieve an overhead of about 20 seconds as shown in Figure 5.24. Obviously, deploying more islands and less workers increases the overhead from 14 seconds by using 5 islands and 20 workers to 22 seconds for the other three combinations averaged over the four considered communication topologies. As shown in Figure 5.25, the container creation time keeps the same trend as the Coarse-Grained Model depicted in Figure 5.20 for the four communication topologies and ranges between 159 seconds and 175 seconds.

For achieving a fair and accurate comparison between the Coarse-Grained - Global Hybrid Model and its two basic parallelization models, namely the Coarse-Grained Model and the Global Model, we compare the fastest execution times of the basic models to the slowest execution time of the Coarse-Grained - Global Hybrid Model. In other words, we consider the Coarse-Grained - Global Hybrid Model with 5 islands - 20 workers per island and Ring topology with the Coarse-Grained Model with 20 islands and Complete topology and the Global Model with 20 workers.

The comparison results show that the Coarse-Grained - Global Hybrid Model with the lowest execution time can achieve a gain in the execution time of about 60% and 70% compared to the Coarse-Grained Model and the Global Model, respectively as shown in

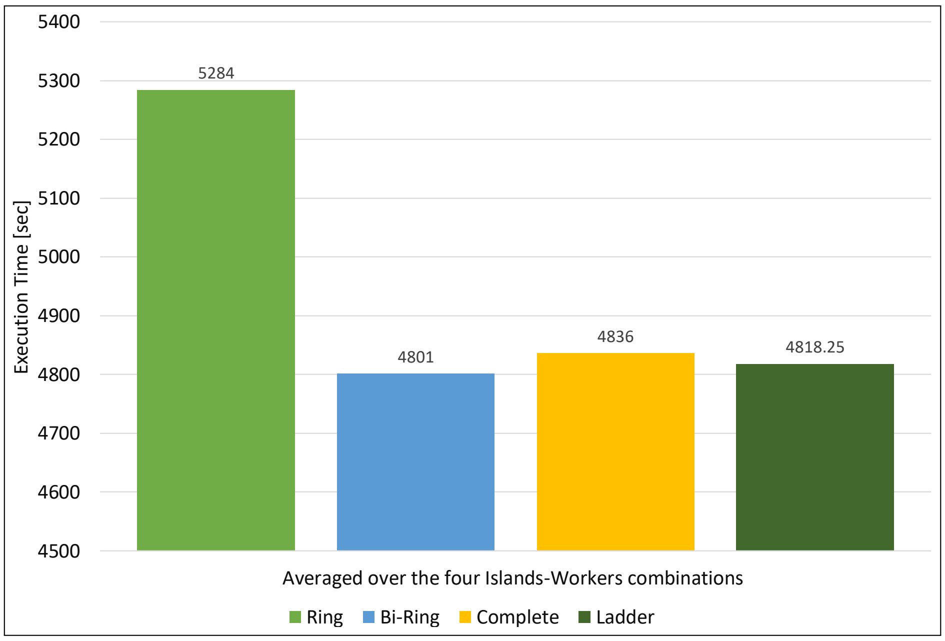


**Figure 5.23.:** Execution time of the Coarse-Grained - Global Hybrid Model for each communication topology averaged over the four applied islands-workers combinations

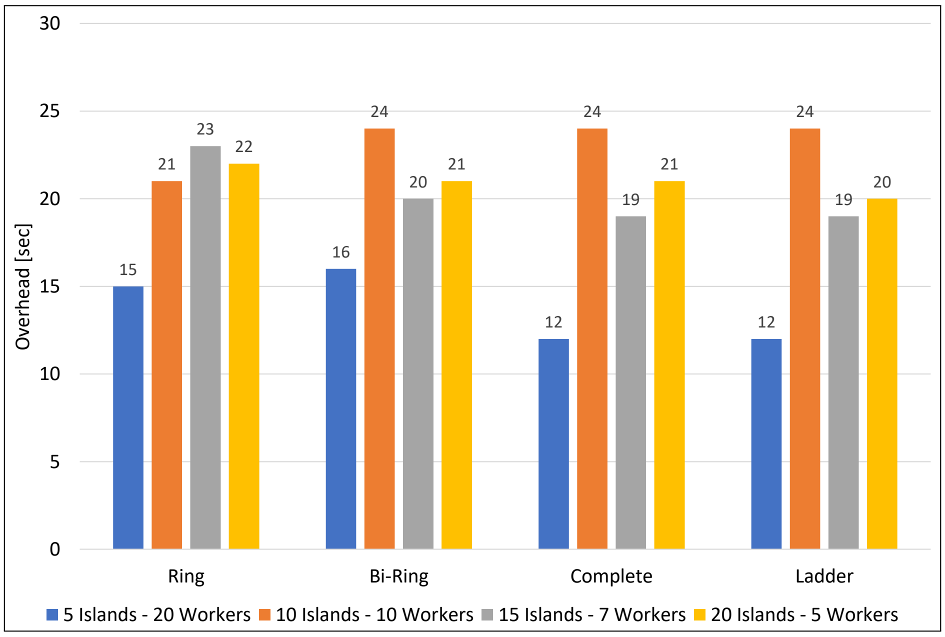


**Figure 5.24.:** Framework overhead of the Coarse-Grained - Global Hybrid Model

Figure 5.26. If we consider the Coarse-Grained - Global Hybrid Model with the fastest execution time, i.e., by using 20 islands - 5 workers per island and Bi-Ring topology, more gains in the execution time of about 10% and 7% can be observed as shown in Figure 5.27.

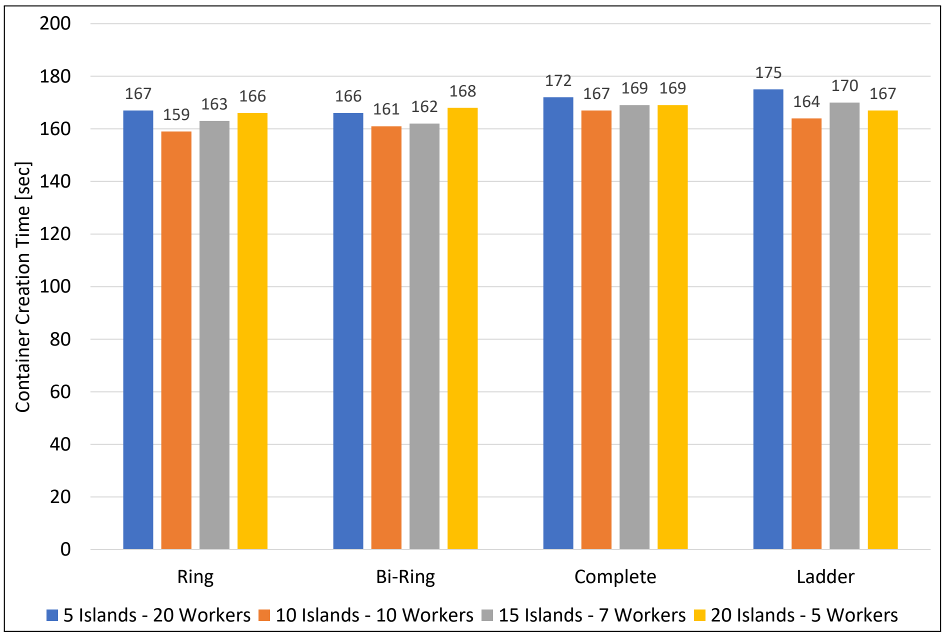


**Figure 5.25.:** Container Creation Time of the Coarse-Grained - Global Hybrid Model

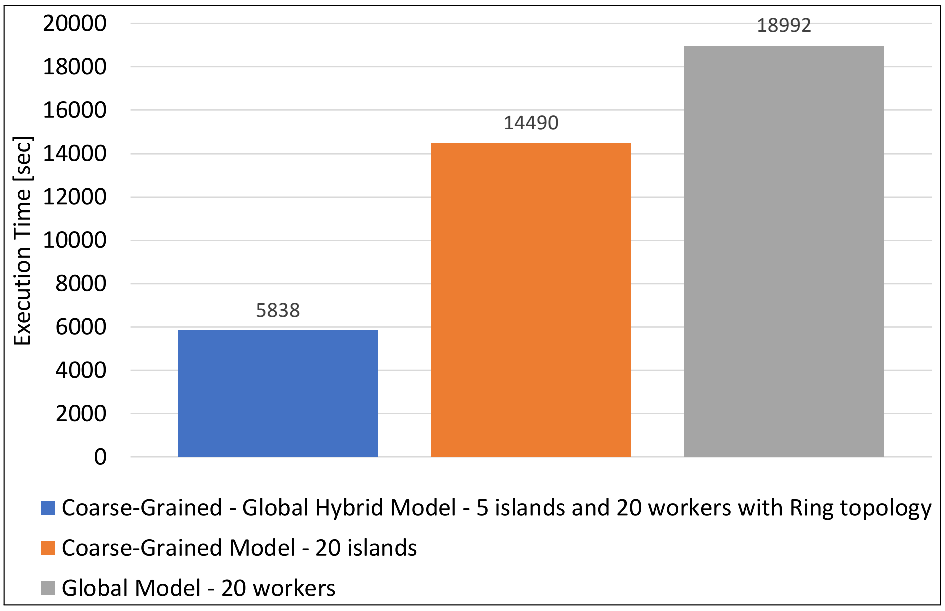


**Figure 5.26.:** Comparing the fastest execution times of the Global and Coarse-Grained models to the slowest execution time of the Coarse-Grained - Global Hybrid Model

The promising results obtained with the Coarse-Grained - Global Hybrid Model can be attributed to the fact that the combination of two levels of parallelization in a hierarchical form allows the use of the advantages of each model. For example, by combining more islands and fewer workers, namely 20 islands and 5 workers in the Coarse-Grained - Global Hybrid Model, we prevent the problem of premature convergence, guarantee an additional acceleration and avoid the bottleneck problem of the Global Model. It is noticeable that not only a very good acceleration is achieved, but also the overhead and the container creation

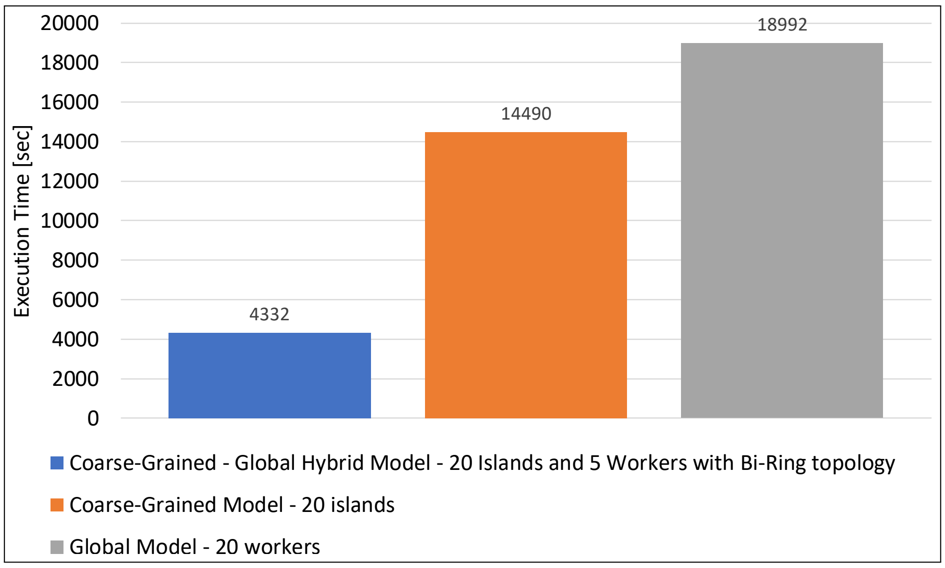


**Figure 5.27.:** Comparing the fastest execution times of the Global and Coarse-Grained models to the fastest execution time of the Coarse-Grained - Global Hybrid Model

time remained almost the same, as for the Global Model and the Coarse-Grained Model, as shown in Figure 5.28.

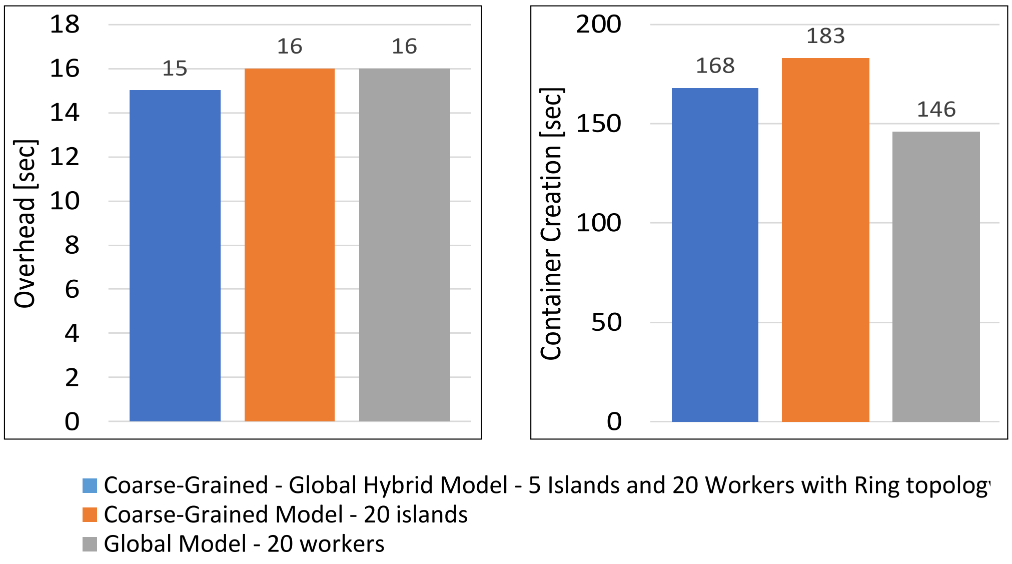


**Figure 5.28.:** Comparing framework overhead and container creation time of the Global and Coarse-Grained models to the Coarse-Grained - Global Hybrid Model

In the last step of this evaluation, we study the effect of applying heterogeneous EA configuration where each island gets different configurations for applying the genetic operators and asynchronous migration policy on the execution time aiming at achieving more acceleration. To this end, the execution time of the Coarse-Grained - Global Hybrid Model, i.e., the one with 20 islands - 5 workers per island and a Bi-Ring topology is chosen as the comparative value. As shown in Figure 5.29, an additional acceleration compared to the synchronous migration policy and homogeneous EA configuration is obtained. The main reason behind such enhancement is that by applying the asynchronous migration policy, the islands continue the evolution process as soon as they receive the migrants without the need to wait for the other islands to complete their current jobs. Moreover, the heterogeneous EA configuration allows better exploitation of the search space. This is due to the fact that heterogeneous configurations enable some islands to explore new areas in the search space while the other islands exploit their already found solutions – if they are promising ones –. This gives GLEAM the ability to find the optimal solution or the near-optimal one in a shorter time. Moreover, if each island has its own settings and configurations, the flexibility to fit the settings of each one to the cluster resources is increased enabling optimal exploitation of the available underlying computing infrastructure.

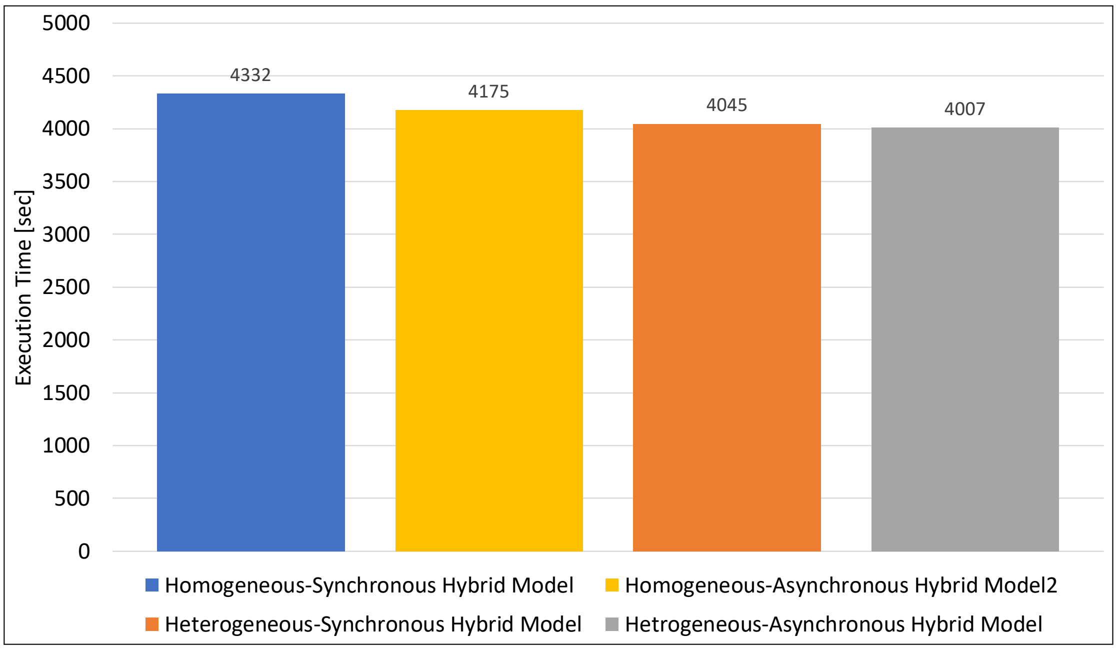


**Figure 5.29.:** Comparing execution time of Homogeneous-Synchronous Hybrid Model to Heterogeneous-Synchronous Hybrid Model and Heterogeneous-Asynchronous Hybrid Model

## 5.3. Summary

In this chapter, BeeNestOpt.IAI is extended to support the Hierarchical Model. It is able to combine two or more models from the basic parallelization models of EAs with moderate implementation efforts, while providing maximum flexibility in mapping optimization tasks to external simulators. To this end, the flexibility, generality and modularity of using microservices and container virtualization to parallelize EAs according to the Hierarchical Model is demonstrated by mapping the Coarse-Grained - Global Hybrid Model to the microservices into the Container Layer as explained in Section 5.1. Combining the Coarse-Grained Model with the Global Model aims at achieving further flexible parallelization by the Global Model and reducing the risk of premature convergence by using the Coarse-Grained Model. In Section 5.1.2, we introduced the execution workflow of the Coarse-Grained - Global Hybrid Model. For the evaluation of the applicability of the proposed approach, a real-world optimization problem is chosen, namely scheduling Distributed Energy Resources (DERs) which is explained in Section 5.2.1. Through such an optimization problem, we are able to evaluate BeeNestOpt.IAI with a practical application. The EA GLEAM with its flexible coding for solving the problem of scheduling DERs is introduced in Section 5.2.3. We deployed the microservices on a cluster with 60 cores, 196 GB of RAM and a 10 GBit/s LAN. We chose a global population of 512 chromosomes and a solution-stop criterion is selected as a global termination criterion. In Section 5.2.4, we evaluate our approach in terms of scalability, overhead and container creation time of BeeNestOpt.IAI for the Coarse-Grained - Global Hybrid Model. In the first step, we evaluated BeeNestOpt.IAI without using any parallelization models, i.e., as a sequential EA for solving the problem of scheduling DERs. Then, the Global Model is evaluated by varying the number of workers between 5, 10, 15 and 20 workers. Afterwards, the Coarse-Grained Model is evaluated by changing the number of islands between 5, 10, 15 and 20 islands for four communication topologies, namely Ring, Bi-Ring, Ladder and Complete. As a migration rate, 10% of the deme size is chosen. Moreover, a synchronous migration policy and a homogenous EA configuration are applied. In the next step, we evaluated the Coarse-Grained - Global Hybrid Model with the same configuration applied to the Coarse-Grained Model while varying the number of islands and workers as follows: 5 islands and 20 workers, 10 islands and 10 workers, 15 islands and 7 workers, 20 islands and 5 workers. Finally, we evaluate the Coarse-Grained - Global Hybrid Model with the asynchronous migration policy and heterogeneous configurations.

| | **EA Sequential Execution ($t_{sequential}$)** | **EA Parallel Execution ($t_{parallel}$)** | | | |
|---|---|---|---|---|---|
| | | Global Model | Coarse-Grained Model | Coarse-Grained - Global Hybrid Model | |
| | | | | Homo. Synch. | Hetero. Asynch. |
| **Execution Time [Seconds]** | 300522 | 18992 | 14490 | 4332 | 4007 |
| **Speedup [$t_{sequential}$/ $t_{parallel}$]** | 1 | 16 | 21 | 69 | 75 |

**Table 5.7.:** Gained speedup through parallelization for solving the problem of scheduling DERs

The concluded results show that the evaluated parallelization models can achieve a considerable speedup ranging between 16 and 75 times as shown in Table 5.7 which is based on the fastest execution time of the two basic parallelization models and the Hierarchical Model. The Coarse-Grained - Global Hybrid Model with asynchronous and heterogeneous configurations is the fastest model. It is faster than the Global Model, the Coarse-Grained

Model and the sequential execution of the EA GLEAM with about four (18992/4007), three (14490/4007) and 75 (300522/4007) times, respectively. The overhead and the container creation time are comparable to the Coarse-Grained Model and the Global Model. With the heterogeneous configurations and the asynchronous migration we achieved a further acceleration of the evolution process. Consequently, BeeNestOpt.IAI is an applicable method to support the Hierarchical Model for solving complex real-world optimization problem, e.g., multi-objective non-linear problems based on a parallel EA with enhanced calculation performance in cluster computing environments.

# 6. Increasing the Applicability of Evolutionary Algorithms in Cluster Computing Environments

EAs are general optimization problem solvers that can be easily applied to various real-world optimization problems. However, EAs perform poorly in solving some complex practical applications. This is due to the fact that EAs suffer from a lack of domain-specific knowledge, the risk of premature convergence to the same suboptimal solution and are computationally very expensive. These drawbacks hinder the common applicability of EAs on a wide range of complex and large-scale optimization problems. It is ambitious to formulate a universal optimization algorithm which is confirmed by the 'No Free Lunch Theorem' proposed by Wolpert and Macready [188]. They stated that there is a little reason to expect a uniform algorithm for solving all classes of optimization problems without some kind of problems in performance or solution quality. In other words, any gained performance on one class of optimization problems is offset by losses in performance on other classes of optimization problems. For enhancing the performance of EAs for a given application scenario, the concept of Hybridization, i.e., combining EAs with other algorithmic approaches into one solution approach is introduced overcoming the above-mentioned drawbacks. The main idea behind the hybridization of different algorithms is to combine methods, techniques and algorithms to solve a complex problem better than using each method separately. Despite EAs being robust general optimization problem solvers, they lack the ability to solve complex real-world optimization problems in a timely manner. Therefore, various research works are concluded to enhance EAs by combining them with other algorithms and/or by integrating domain-specific knowledge into them. The results show that improvements in the form of better solution quality, faster convergence speed and better exploration and exploitation of the solution space, to name a few, can be obtained (see, Section 3.2 for more details).

Parts of this chapter are reproduced from:

- Khalloof, H., Ciftci, S., Shahoud, S., Duepmeier, C., Foerderer, K., & Hagenmeyer, V. "Facilitating the hybridization of parallel evolutionary algorithms in cluster computing environments". In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. (2022), pp. 2001–2008. DOI: 10.1145/3520304.3533997

To guarantee a seamless and effective hybridization process, two main aspects have to be considered, namely an efficient hybridization concept and a software solution that facilitates such a hybridization at any part of EAs. In this chapter, the main focus is on introducing a modular, extendable, flexible, generic and scalable software solution for hybridizing EAs in cluster computing environments to answer the research question **[RQ5]**. The generic software architecture of BeeNestOpt.IAI is extended to support the hybridization of EAs. In other words, a new microservice called ‘Support-and-Learning Service (S&L Service)’ is added to the software architecture for facilitating the integration of any hybridization approach into any step of EA. Furthermore, a new methodology to hybridize EAs with ML techniques in high-performance computing environments is introduced. The generality of the proposed software solution is tested by seeding the initial population of EAs either with prior solutions or with new ones generated based on the extracted knowledge from the already found solutions. To achieve that, two ML techniques, namely clustering and sequence-to-sequence Long Short-Term Memory (LSTM) with multiple-output layers [21] are used. For demonstrating the beneficial traits of the proposed ML hybridization approach, the use case scenario introduced in Section 5.2.1 for scheduling Distributed Energy Resources (DERs) is applied. The scheduling task is formulated as a combinatorial non-convex mixed-integer multi-objective optimization problem where 50 DERs are utilized with two consumption profiles.

Section 6.1 covers the extension of the generic microservice-based architecture of BeeNestOpt.IAI to support the hybridization process at any part of EAs in high-performance computing environments. In Section 6.2, the newly proposed ML hybridization approaches, namely the unsupervised hybridization approach and the supervised hybridization approach to assist EAs by generating the initial population are introduced. In Section 6.3, the evaluation concept, the realization of both ML hybridization approaches to assist the parallelized EA GLEAM according to the Global Model and the first obtained results by deploying the proposed hybridization solutions on a cluster for solving the problem of scheduling DERs are presented. Section 6.4 concludes this chapter. The research contributions presented in this chapter are the main topics of our paper [104].

## 6.1. BeeNestOpt.IAI for Hybridization of Evolutionary Algorithms

For facilitating the hybridization of already developed EAs – that they support some kind of hybridization but not in cluster computing environments – with any algorithmic approaches in high-performance computing environments, a generic, modular, flexible and scalable software architecture is required. This architecture has to enable easy integration of EAs and hybridizes them with any method or algorithm with minimum adaptation efforts. Microservice-based software architecture provides such flexibility by introducing interfaces for the integration of any method that can assist EAs. Furthermore, it allows EAs to access the hybridization methods and integrate them, e.g., in the generation step of the initial population. To this end, BeeNestOpt.IAI is adapted to support the hybridization of EAs by adding a new microservice called ‘Support-and-Learning Service’ (S&L Service) which is shown in Figure 6.1. It provides the ability to plug in any hybridization model, e.g., ML model to assist EAs by solving real-time complex and large-scale optimization problems. The S&L Service supports the publish/subscribe messaging paradigm to reduce the coupling between it and other services, enabling the S&L Service to communicate with any service of BeeNestOpt.IAI. Hence, the S&L Service allows effortless hybridization of any part of EAs. For example, the S&L Service can communicate with the EA Service to assist it by generating the initial population or with the Ca. Service for estimating the fitness function. Furthermore, the publish/subscribe messaging paradigm guarantees an efficient and reliable data exchange between the S&L Service and the others.

To achieve the above functionalities, the S&L Service defines an I/O adapter to link any hybridization model, e.g., ML model to BeeNestOpt.IAI. The I/O adapter, e.g., glue code receives hybridization requests sent from other services and extracts the information needed by the considered hybridization model. After processing each hybridization request by the deployed hybridization model, the I/O adapter transfers the results into a form that can be consumed by other services. Data exchange between the hybridization model and the adapter is achieved through file access. The I/O adapter writes the extracted information into a file that is read by the hybridization model which applies the configuration, processes the request and writes the result into another file as output to be read by the I/O adapter as shown in Figure 6.1. We leverage the container technologies to enable seamless deployment of the S&L Service in a high-performance computing environment like a cluster. In other words, the S&L Service can be deployed as often as needed, as long as enough computation resources can be provided.

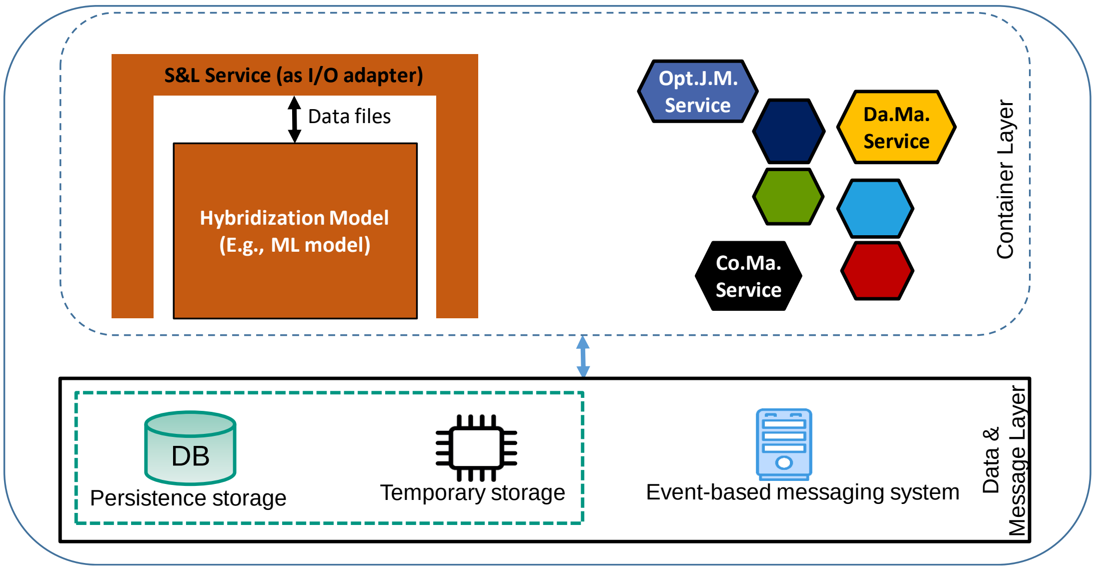


**Figure 6.1.:** Support-and-Learning Service (S&L Service) in the Container layer

To communicate with other services in the Container Layer, the S&L Service provides two publish/subscribe channels as shown in Table 6.1. While the first channel is used by other services to publish hybridization requests including, e.g., hybridization model – supervised

| Method | Address | Description |
|---|---|---|
| SUBSCRIBE | hybridization.request. job.{job-id}.service. {serviceName} | Get hybridization requests including configurations like hybridization model, ML algorithm, data, etc. from other services and applications. |
| PUBLISH | hybridization.result. job.{job-id} | Publish the results, e.g., seeds. |

**Table 6.1.:** Publish/subscribe channels of the S&L service

or unsupervised – the latter one is used by the S&L Service for publishing the results of processing hybridization requests.

## 6.2. Machine Learning-based Approaches for Hybridizing Evolutionary Algorithms

Generally, EAs discard domain-specific knowledge from evolution and start from scratch for each optimization problem. Therefore, two hybridization approaches based on the well-known ML techniques, namely unsupervised and supervised learning to utilize such knowledge for assisting EAs in generating the initial population are proposed as shown in Figure 6.2. Precisely, they exploit the domain-specific knowledge and the prior solutions that are already found for similar problems for seeding the initial population and guiding the search process of the EAs. Based on only domain-specific knowledge, the unsupervised hybridization approach selects the nearest similar solutions from past runs of EAs as they exist and seed them in the initial population. However, the supervised hybridization approach instruments a learner to learn the mapping between the domain-specific knowledge and the prior solutions to generate completely new solutions, i.e., seeds.

Domain-specific knowledge can be generally grouped into two groups, namely the one that is directly used by EAs to find the solutions and the one that is indirectly related to the found solutions. For example, an EA solves the problem of unit commitment by computing an adequate day-ahead scheduling plan for a customer with the minimum cost as an objective. The EA requires the predicted load profile of the customer, the predicted energy generation for the considered time interval for each energy resource and its corresponding price. This data is the direct knowledge used by EAs to assess the fitness of each proposed solution. However, the weather data is an example of indirect knowledge that is significantly related to the scheduling plans. As shown in Figure 6.2, the direct and indirect knowledge related to the obtained solutions is stored in the ‘Prior Knowledge Repository’. This knowledge is known as features and they are used to describe the solutions saved in another repository, namely the ‘Prior Solutions Repository’. While the unsupervised hybridization approach uses these features to select the nearest similar solutions from the ‘Prior Solutions Repository’, the supervised hybridization approach learns how to map the prior solutions to these features.

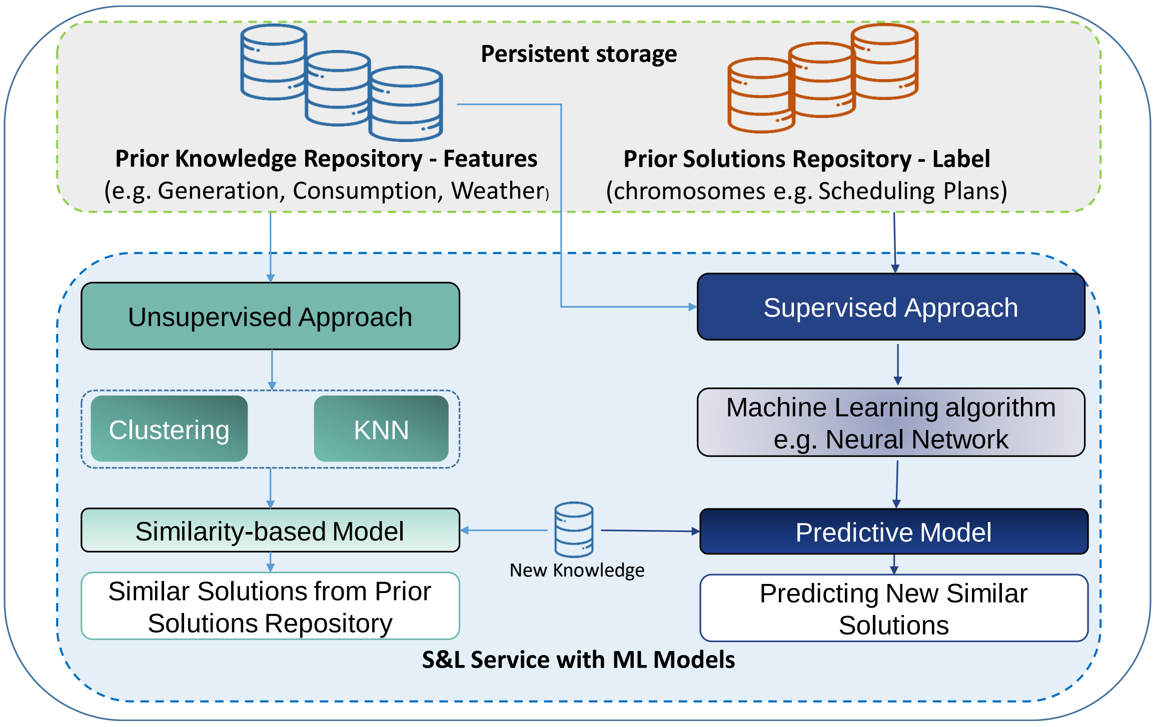

Persistent storage
Prior Knowledge Repository - Features
(e.g. Generation, Consumption, Weather)
Prior Solutions Repository - Label
(chromosomes e.g. Scheduling Plans)
Unsupervised Approach
Clustering
KNN
Similarity-based Model
Similar Solutions from Prior Solutions Repository
New Knowledge
Supervised Approach
Machine Learning algorithm e.g. Neural Network
Predictive Model
Predicting New Similar Solutions
S&L Service with ML Models

**Figure 6.2.:** Concept of the proposed ML-based hybridization approaches of EAs

Generally, the S&L Service needs two input parameters defined by the user, namely 'mode' and 'featureType'. The 'mode' parameter is used to select the method or the technique, namely, the ML technique that is used to hybridize EAs, i.e., supervised or unsupervised. The 'featureType' parameter is used to define which features from the 'Prior Knowledge Repository' are selected for performing the training process of the ML model.

### 6.2.1. Unsupervised Machine Learning-based Approach for Hybridizing EAs

Unsupervised learning aims to find the regularities in the input data. In other words, it discovers the structure of the input data including their hidden patterns. Clustering is the most famous unsupervised learning technique by which the input data is divided into clusters or groups of similar patterns based on some measures like Euclidean distance [106]. As shown in Figure 6.2, the clustering algorithm is trained with domain-specific knowledge, i.e., features that correspond to the solutions obtained from previous runs of the used EA. It groups similar features based on a similarity measure into several clusters. When a new problem of the same domain needs to be solved, the clustering model assigns the new unseen features to the closest cluster. Then, the proposed approach returns one or multiple potential similar solutions corresponding to the features within the chosen cluster from the 'Prior Solutions Repository'. These solutions are used to enhance EAs by integrating them into the evolutionary process, e.g., into the initial population.

The proposed unsupervised ML approach provides three modes, namely cluster, best and mix. These modes provide an EA with more flexibility for selecting solutions from the returned similar ones to be used in the evolution process. The cluster mode returns a random set of similar solutions out of the cluster, in which the new features are clustered. In contrast, the best mode only returns the nearest similar solutions without considering any cluster borders. Since the diversity of a population plays a key role for EAs, we decided to define a new mode, namely the mix mode to increase the diversity of the returned similar solutions. In this mode, the best mode is applied and some randomness is added to it. Precisely, half of the returned similar solutions are selected according to the best mode and the other half is returned by the cluster mode.

Since the high dimensional input spaces can pose challenges for the clustering algorithm, dimensionality reduction methods such as Principal Component Analysis (PCA) or Autoencoder (AE) can be used to reduce the dimensions of the input space to the optimal or more adequate one aiming at performing better clustering.

### 6.2.2. Supervised Machine Learning-based Approach for Hybridizing EAs

The supervised hybridization approach aims at extracting patterns from domain-specific knowledge and information from prior solutions to predict potential solutions for a new given problem instance. This approach uses a part of the feature space used in the unsupervised hybridization approach combined with the corresponding prior solutions as labels. It trains a learner, e.g., ANN with the labeled data to learn the mapping between the domain-specific knowledge and prior solutions. Hence, the supervised hybridization approach does not return solutions as they exist in the 'Prior Solutions Repository' but generates new solutions based on the learned knowledge. The proposed approach generates solutions that can serve as part of the initial population of an EA. Since the labels are the solutions representing the chromosomes, the learning algorithm has to learn the structure of the chromosome, i.e., the genetic representation. To this end, the prior solutions, i.e., chromosomes are processed and adapted to a usable label format and then combined with the corresponding features. In contrast to the unsupervised hybridization approach, the supervised hybridization approach takes the advantage of learning patterns from the whole repository and exploits them to generate new chromosomes which could be more suitable than the ones returned by the unsupervised hybridization approach. This is due to the fact that the unsupervised hybridization approach can seed the initial population with suboptimal solution. However, the supervised hybridization approach learns from all available solutions, i.e., the optimal and suboptimal ones. An ANN with its design flexibility can be instrumented as a learner to generate the chromosomes with the same genetic representation as the prior ones. In the context of this work, an LSTM is utilized as an implementation of the ANN.

To summarize, the unsupervised hybridization approach serves as a straightforward approach for the hybridization of an EA with ML. Grouping similar solutions to a cluster limits the search process for finding the most similar solutions to several bounded pools of potential solutions. The unsupervised hybridization approach lacks the ability to learn from similar solutions and generate new solutions for a new similar problem instance. In contrast to the unsupervised hybridization approach, the supervised hybridization approach learns from the prior solutions and generates new solutions increasing the diversity of the initial population. The main drawback of this approach is how to adapt the historical solution to a useful form to be used as labels. Moreover, designing an ML algorithm to learn the mapping between features and complex format of labels represents a challenge.

## 6.3. Evaluation

In this evaluation, we investigate how the proposed microservice-based solution facilitates the hybridization process of EA. Furthermore, we study and analyze the effect of seeding the initial population of an EA on the convergence speed and solution quality. To achieve that, the unsupervised and supervised hybridization approaches are realized and integrated into the S&L Service to assist EA by generating the initial population for solving the problem of scheduling Distributed Energy Resources (DERs). The use case scenario introduced in Section 5.2.1 for scheduling 50 DERs is considered where the optimization problem is formulated as a combinatorial non-convex mixed-integer multi-objective problem. The EA GLEAM with its genetic representation introduced in Sections 2.2.3 and 5.2.3 is instrumented to solve this problem. Both approaches can be combined with any other EA which supports the hybridization type of seeding the initial population. For realizing both hybridization approaches for scheduling DERs, the S&L Service introduces an additional input parameter, namely ‘date’ which has the format ‘YYYY-MM-DD’ representing the requested date to be scheduled. Based on knowledge-domain data corresponding to this date, the S&L Service returns a set of similar solutions to be seeded into the initial population. The ‘Prior Solutions Repository’ is filled with a list of chromosomes where each one represents an hourly scheduling plan. These plans are calculated in advance for one year for the same use case. Three basic features, namely consumption (c), generation (g) and weather (w) are contained in the ‘Prior Knowledge Repository’. These features correspond to the scheduling plans stored in the ‘Prior Solutions Repository’. The consumption and generation features are extracted from the solar home electricity dataset provided by Ausgrid [153] which is explained in Section 5.2.2.

For defining the consumption behavior of the aggregated load (house symbol located in the blue DER of Figure 5.4), real-world load profiles of a customer from the Ausgrid dataset are selected. For the power generation process, 50 DERs from the Ausgrid dataset based on their distance to the considered customer are selected. For the weather feature, the historical weather data provided by the Bureau of Meteorology  is used. This data contains weather information collected from 153 weather stations located in the region of New South Wales where the Ausgrid dataset is collected. The weather data of the weather station Observatory Hill is used, since it is located in the center of Sydney with relative proximity to the selected

http://www.bom.gov.au/ (Accessed: 10.02.2023)

| Weather Feature | | Description | Units |
|---|---|---|---|
| Temperatur | Min | Minimum temperature in the 24 hours at 9:00. | degree celsius |
| | Max | Maximum temperature in the 24 hours at 9:00. | degree celsius |
| Rain | | Precipitation (rainfall) in the 24 hours at 9:00. | millimeters |
| Evapotranspiration | | "Class A" pan evaporation in the 24 hours at 9:00. | millimeters |
| Solar Radiation | | Power per unit area received from the Sun. | watt per square meter (W/m2) |
| Average wind speed | | Wind speed measure over 10 meters. | Meters per second |
| Relative humidity | Min | Minimum relative humidity in the 24 hours at 9:00. | percent |
| | Max | Maximum relative humidity in the 24 hours at 9:00. | percent |

**Table 6.2.:** Explanation of the measured weather values of the weather station Observatory Hill – Australia

customer and most of the selected DERs. The Australian Bureau of Meteorology does not freely provide the weather data in a fine-grained resolution, e.g., 15-minutes or 30-minutes but only in the form of a summary for each day, e.g., daily resolution as shown in Table 6.2.

### 6.3.1. Unsupervised Approach for Scheduling Distributed Energy Resources

The goal of the unsupervised approach is to return a set of scheduling plans from the ‘Prior solutions Repository’ based on the similarity metric. To achieve that, the unsupervised approach builds a clustering model that groups similar domain-specific knowledge stored in the ‘Prior Knowledge Repository’ together into one cluster and implicit groups the scheduling plans which are corresponded to the clusters together as shown in Figure 6.3.

For the considered datasets, we have access to the hourly generation of each DER, the hourly consumption of the customer and weather data with a daily resolution. These three feature sets represent the knowledge space stored in the ‘Prior Knowledge Repository’. The consumption data of the customer plays a key role in the scheduling process performed by GLEAM and therefore is essential in the clustering process. It is used to cluster consumption profiles into well-separated clusters based on their similarity. For example, by applying the distance metric to find the similarity between two profiles, the absolute difference between two consumption profiles is calculated by subtracting the hourly consumption values from each other. Then, the sum of the absolute differences is calculated where is a low value indicating high similarity. For the generation feature, only power generation between 9

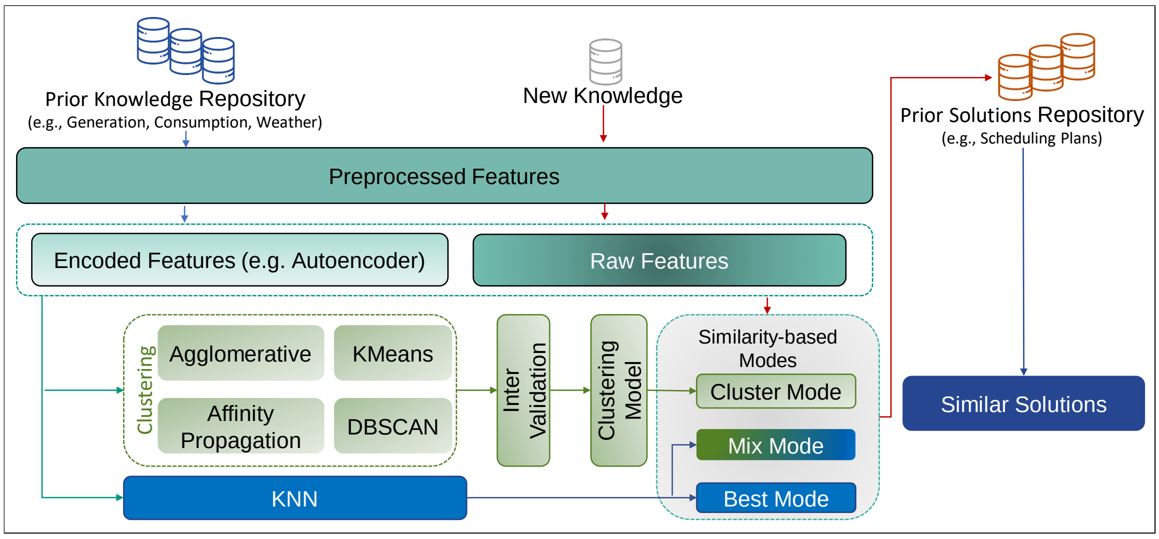


**Figure 6.3.:** Concept of the unsupervised approach to assist EAs for scheduling DERs

o'clock and 17 o'clock, i.e., the main PV hours, is taken into account. This is due to the fact that the used DERs are mainly based on PVs for generating power as explained in Section 5.2.1. Both features are directly involved in the process of creating scheduling plans, since they are essential for calculating the objective functions introduced in Section 5.2.1. Since power consumption and power generation are influenced by the weather, the weather data is considered as an indirect factor that is involved in the process of creating scheduling plans. For example, the maximum temperature and the solar radiation have a significant impact on the generated energy as illustrated in Figures 6.4 and 6.5. In the 'Prior Knowledge Repository', there are 24 features for hourly power consumption profiles of a customer, nine features representing hourly power generation aggregated over the selected 50 DERs and eight features for the weather data. Besides the three feature sets, four combinations between them are defined, namely consumption-generation (cg), consumption-weather (cw), generation-weather (gw) and consumption-generation-weather (cgw). This results in seven feature sets, i.e., c, g, w, cg, cw, gw and cgw which are considered for the clustering process.

For building an adequate clustering model, four clustering algorithms, namely KMeans, Affinity Propagation, Agglomerative Clustering and DBSCAN are used. These algorithms follow different methodologies and use different parameter settings to cluster the input. The adequate values for such parameters for the four algorithms are not known a priori. Therefore and to identify which clustering algorithm performs better than the others, we test each clustering algorithm with different parameter settings and various cluster sizes. The algorithms are compared to each other considering three clustering criteria, namely the Silhouette Score [160], the Dunn Index [47] and the Connectivity Score [78]. These criteria are used to measure the performance of a cluster algorithm indicating how well the clustering algorithm has performed on a given set of features. Raw features are more challenging for all clustering algorithms. This is due to the fact that the high dimensionality features lead to inaccurate similarity computation. Based on visual cluster analysis, the

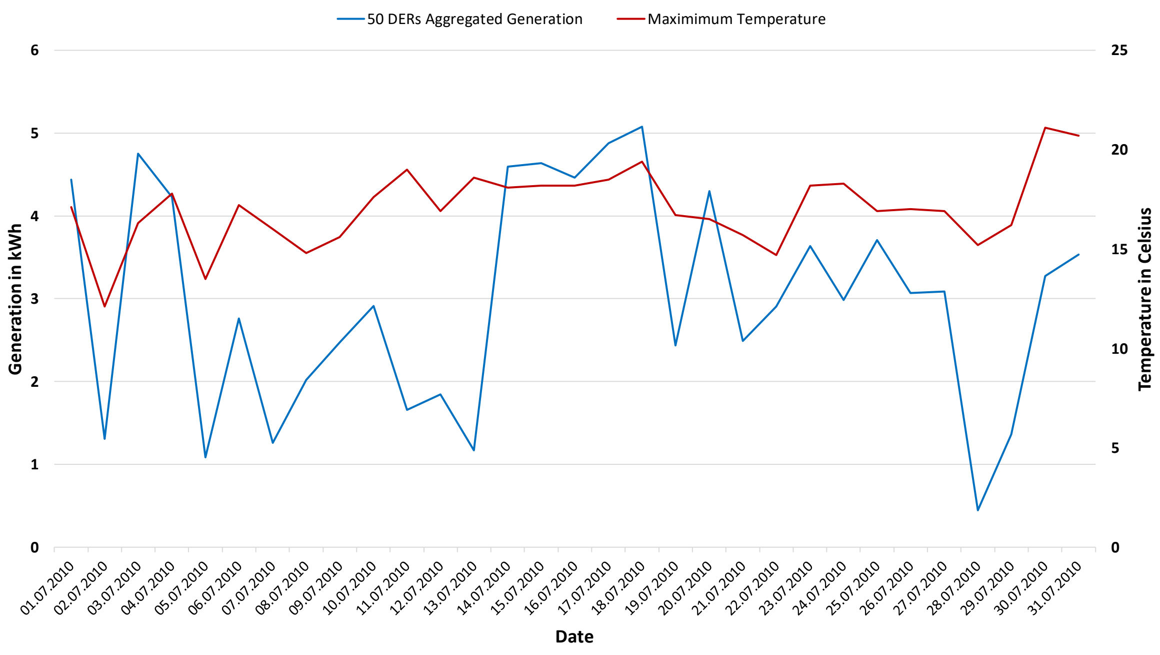


**Figure 6.4.:** Blue line depicts the 50 DERs stacked generation during the first month of the training dataset. The red line depicts the corresponding temperature measured on those days

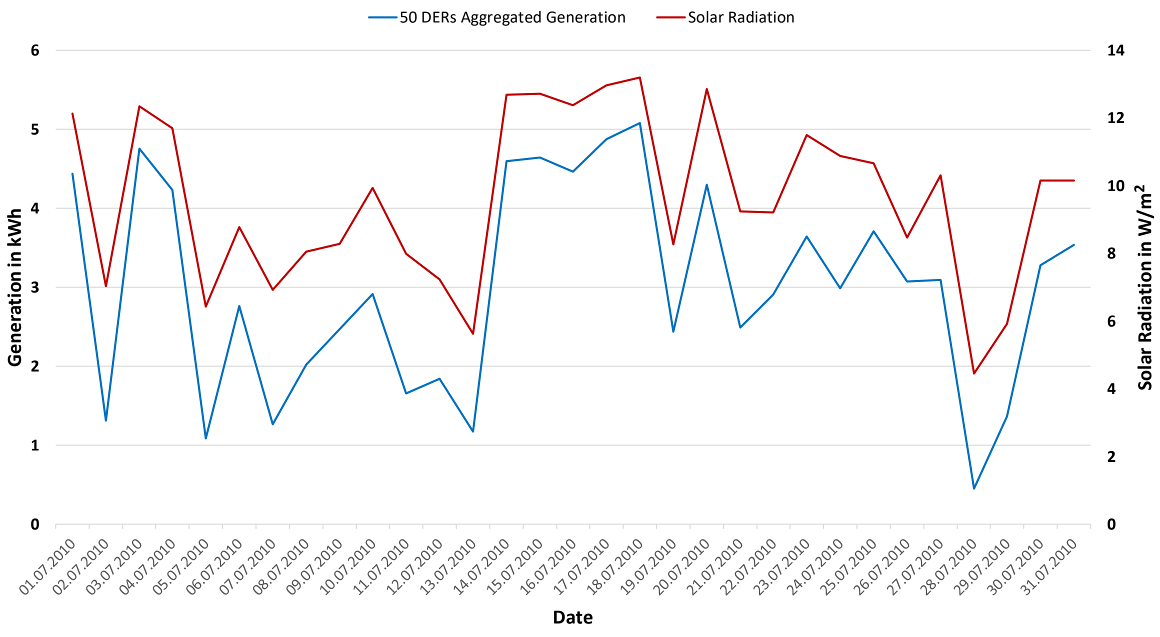


**Figure 6.5.:** Blue line depicts the 50 DERs stacked generation during the first month of the training dataset. The red line depicts the corresponding solar radiation measured on those days

best-performing cluster algorithm for the raw features of the sets is the Agglomerative Cluster algorithm with average as a linkage parameter. For performing better clustering results, an autoencoder technique is applied to reduce the representation of each feature set to two features in total. The results show that the clustering algorithms perform better on

| Features | Number of neurons in each layer | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | Reduced representation | | | | |
| Consumption (c) | | 24 | 12 | 6 | 2 | 6 | 12 | 24 | |
| Generation (g) | | | 9 | 6 | 2 | 6 | 9 | | |
| Weather (w) | | | 7 | 4 | 2 | 4 | 7 | | |
| cg | 33 | 24 | 12 | 6 | 2 | 6 | 12 | 24 | 33 |
| cw | 31 | 24 | 12 | 6 | 2 | 6 | 12 | 24 | 31 |
| gw | | 16 | 12 | 6 | 2 | 6 | 12 | 16 | |
| cgw | 40 | 24 | 12 | 6 | 2 | 6 | 12 | 24 | 40 |

**Table 6.3.:** The structure of the used Autoencoders for each feature set combination

two-dimensional data. The number of layers and number of neurons of each autoencoder are selected based on the number of features within each feature set as shown in Table 6.3.

The best-performing clustering algorithm based on the encoded features is the KMeans clustering algorithm with ten clusters. Both clustering results, i.e., based on the raw and encoded features, are visualized in Figures 6.6, 6.7, 6.8, 6.9, 6.10, 6.11 and 6.12. Each figure depicts and compares the clustering results based on the raw and the encoded features. While each figure on the left side illustrates the clustering results of applying the Agglomerative Cluster algorithm with the raw features, the ones on the right side depict the clustering results of applying the KMeans algorithm on the reduced features by using an autoencoder technique. Both algorithms are tested with different numbers of clusters according to the feature type. The results show that the raw features have many overlaps between clusters. In most cases, the clustering algorithm of the raw features tends to cluster most data points in one big cluster and the remaining ones are grouped in small clusters where each one has between 1 and 20 data points. In other words, the clusters of raw features are mostly very small except for one big cluster consisting of 80–95% of all data points. However, the reduced features are clustered more clearly than the raw features and data points are more evenly distributed over all clusters as shown in Figures 6.6, 6.7, 6.8, 6.9, 6.10, 6.11 and 6.12. It is clear that better clustering results and even distribution of data points are obtained by using the reduced feature sets.

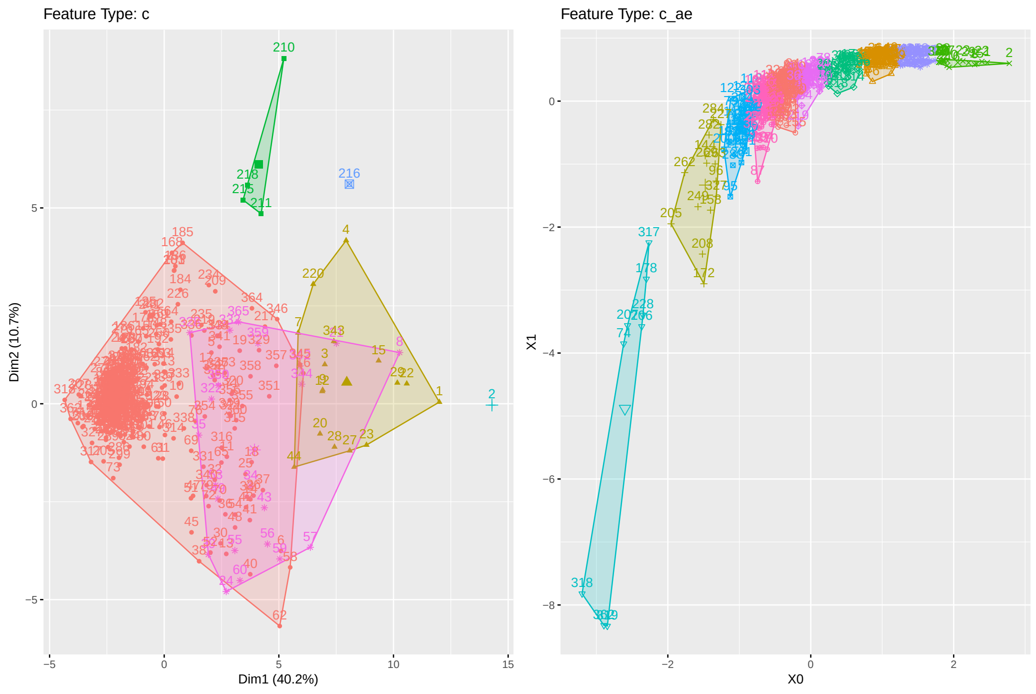


**Figure 6.6.:** Clustering of the Consumption (c) features (24 features). The left side displays the raw values of the c features and on the other side, the same features reduced to two features (X0, X1) by utilizing an Autoencoder (c_ae)

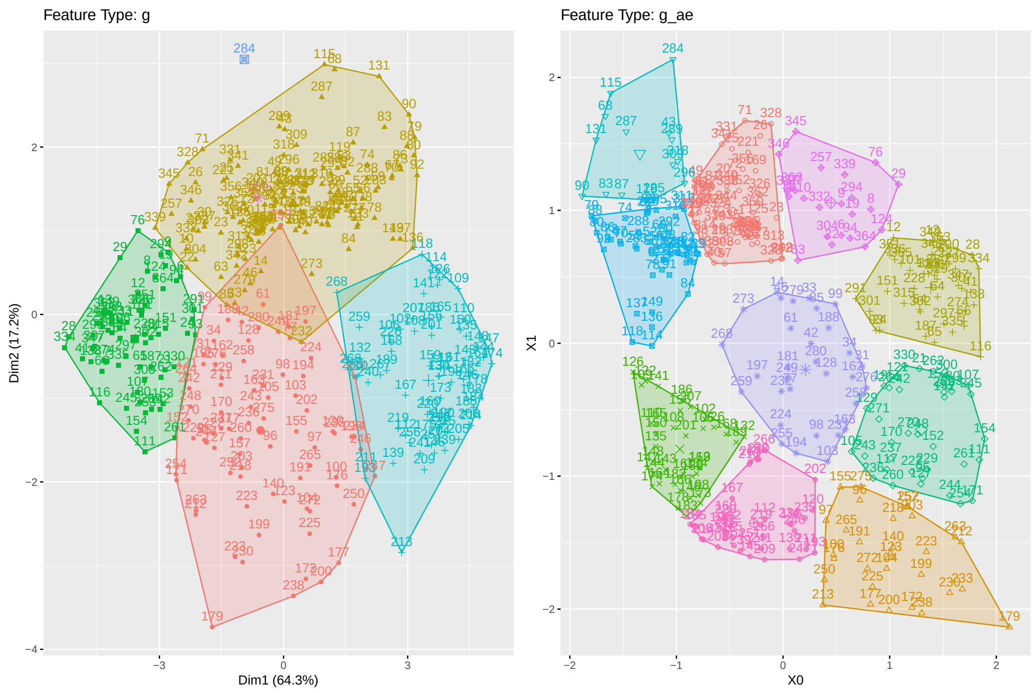


**Figure 6.7.:** Clustering of the Generation (g) features (9 features). The left side displays the raw values of the g features and on the other side, the same features reduced to two features (X0, X1) by utilizing an Autoencoder (g_ae)

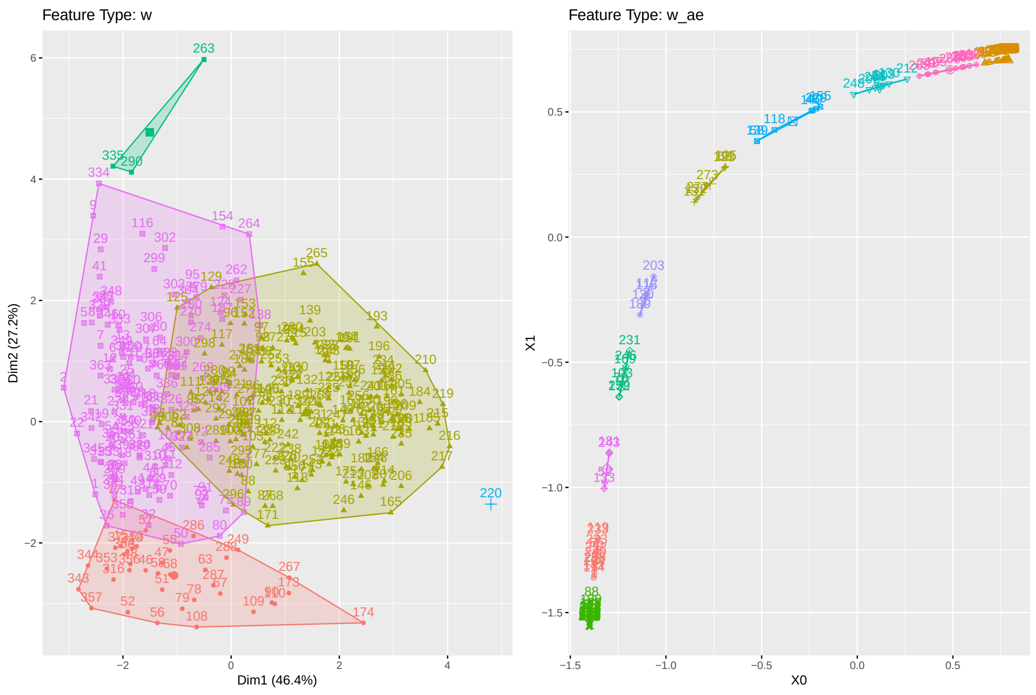


**Figure 6.8.:** Clustering of the Weather (w) features (8 features). The left side displays the raw values of the w features and on the other side, the same features reduced to two features (X0, X1) by utilizing an Autoencoder (w_ae)

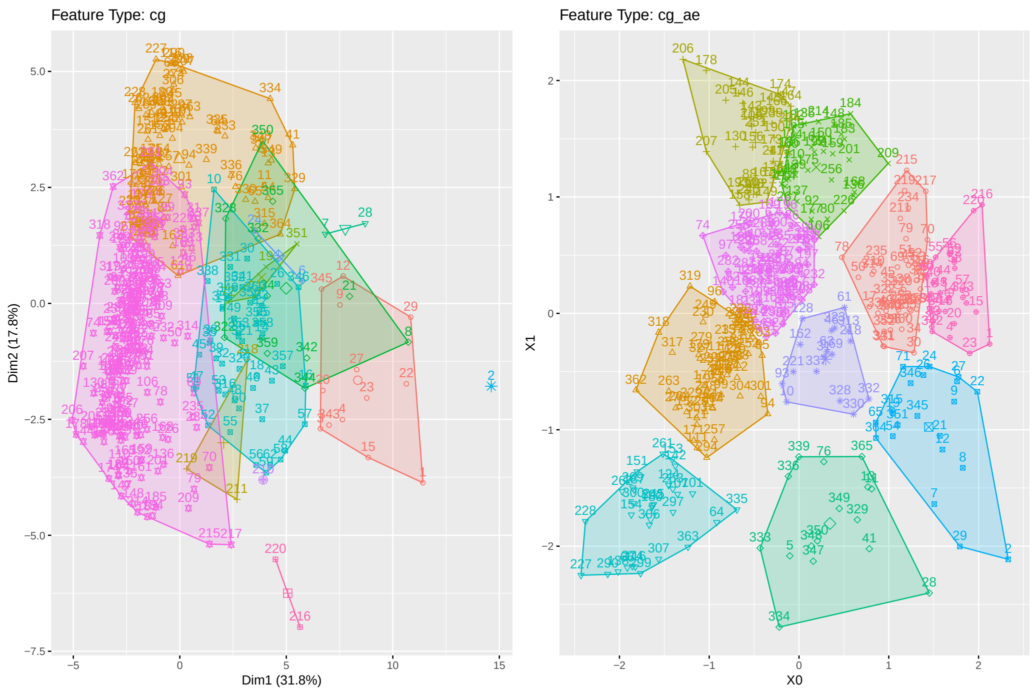


**Figure 6.9.:** Clustering of the combination of Consumption and Generation (cg) features (33 features). The left side displays the raw values of the cg features. On the other side, the same features reduced to two features (X0, X1) by utilizing an Autoencoder (cg_ae)

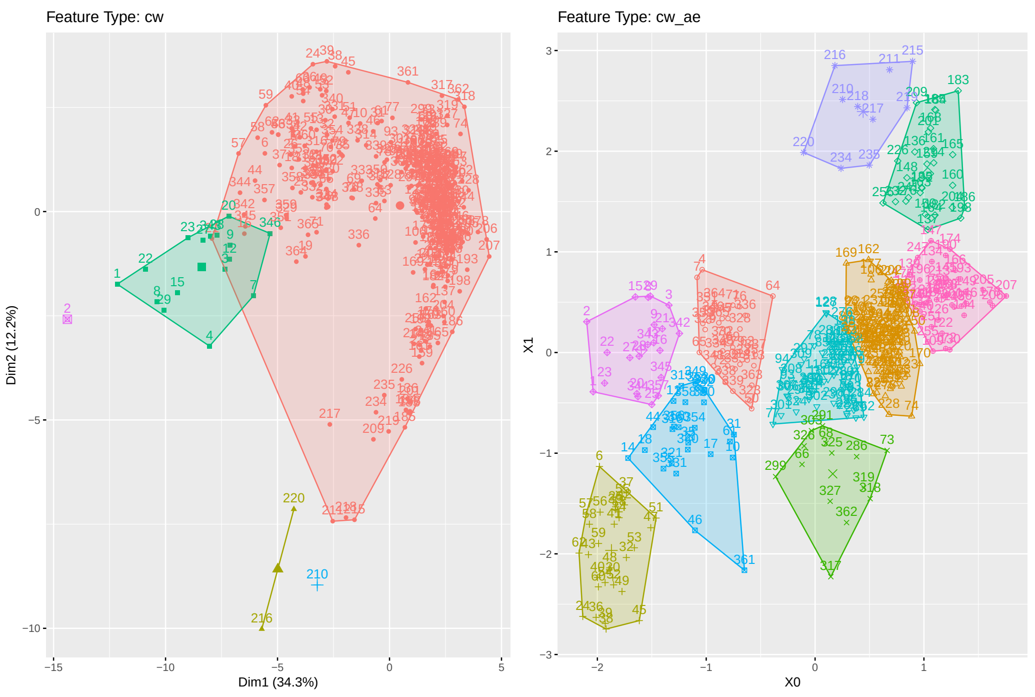


**Figure 6.10.:** Clustering of the combination of Consumption and Weather (cw) features (32 features). The left side displays the raw values of the gw features. On the other side, the same features are reduced to two features (X0, X1) by utilizing an Autoencoder (cw_ae)

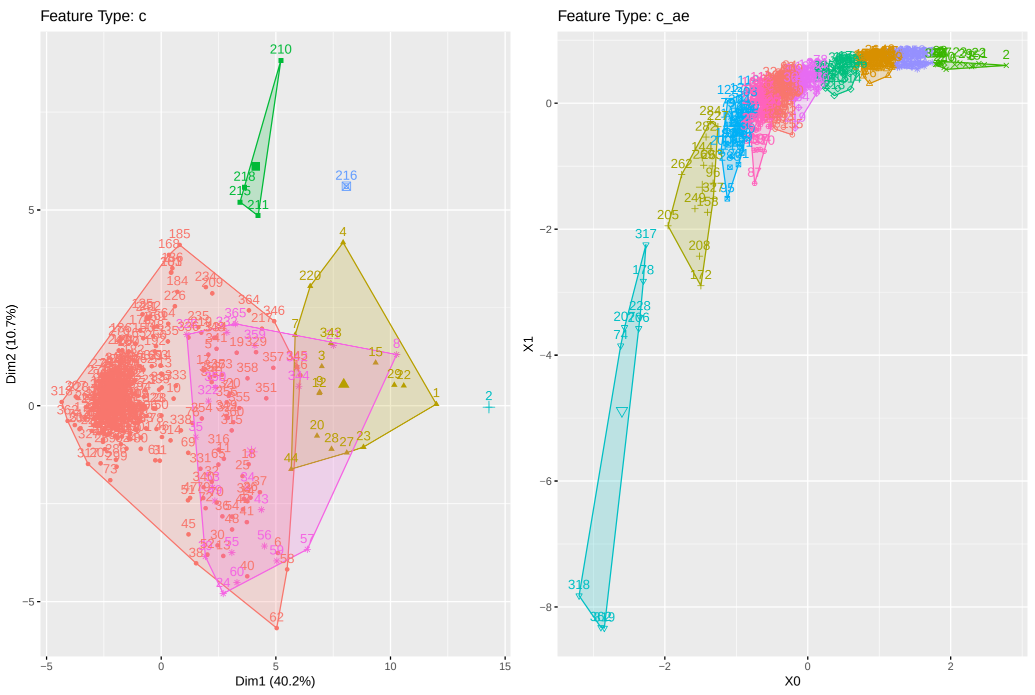


**Figure 6.11.:** Clustering of the combination of Generation and Weather (gw) features (17 features). The left side displays the raw values of the gw features. On the other side, the same features are reduced to two features (X0, X1) by utilizing an Autoencoder (gw_ae)

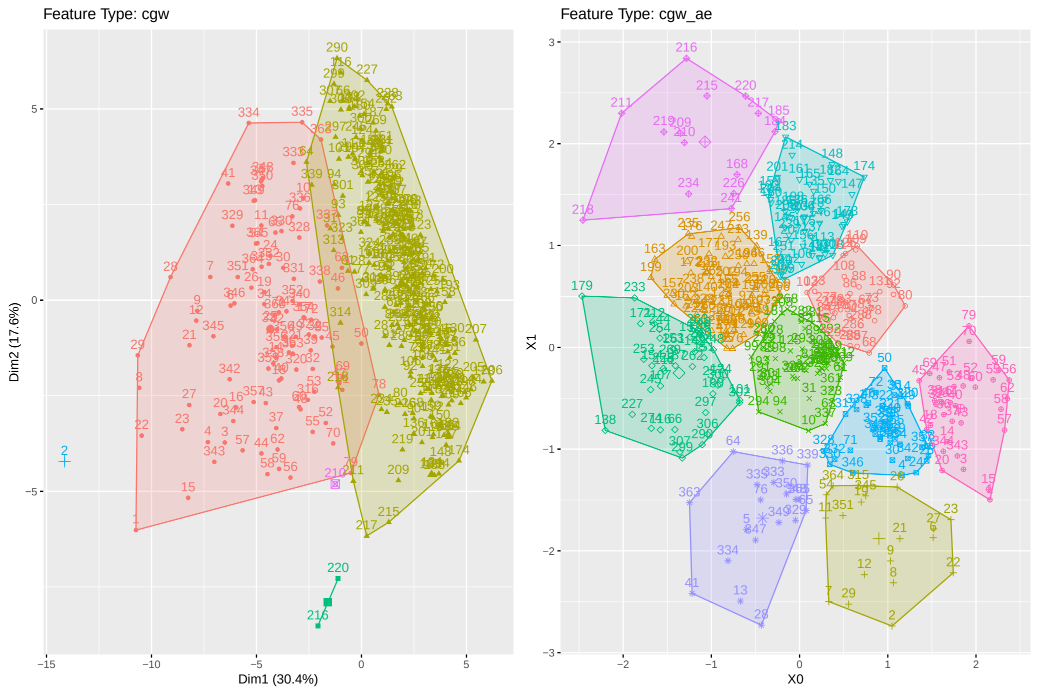


**Figure 6.12.:** Clustering of the combination of Consumption, Generation and Weather (cgw) features (41 features). The left side displays the raw values of the cgw features. On the other side, the same features are reduced to two features (X0, X1) by utilizing an Autoencoder (cgw_ae)

### 6.3.2. Supervised Approach for Scheduling Distributed Energy Resources

The goal of the supervised approach is to create one or multiple new solutions either to be seeded into the initial population of the EA GLEAM or to be used directly as scheduling plans for the customer. To achieve that, a dataset which contains the available feature sets in the 'Prior Knowledge Repository' as features and the already found scheduling plans stored in the 'Prior Solutions Repository' as labels is created as shown in Figure 6.13. Building an ML model that can learn the mapping between the features and the corresponding scheduling plans, i.e., the label is not a trivial task. This is due to the fact that the length of the labels is not fixed as explained in Sections 2.2.3 and 5.2.3. In other words, the EA GLEAM codes each scheduling plan as a chromosome with a variable number of genes where each gene has three discrete variables, namely Unit ID, Start, Duration and one continuous variable, i.e., Power Fraction (PF).

To tackle such challenges, a sequence-to-sequence (Seq2Seq) learning method with multiple outputs [80], namely LSTM is proposed (see, Section 2.4.2). As shown in Figure 6.14, the proposed Seq2Seq LSTM model is composed of an encoder and a decoder. For training a LSTM model, the encoder is designed with $T$ time steps – e.g., 24 if an hourly day-ahead scheduling plan is required – to capture the context of the input. For each time step, the input is provided in the form of a vector containing the hourly consumption of a customer and the generation for $N$ DERs – e.g., 50 DERs for the use case scenario introduced in

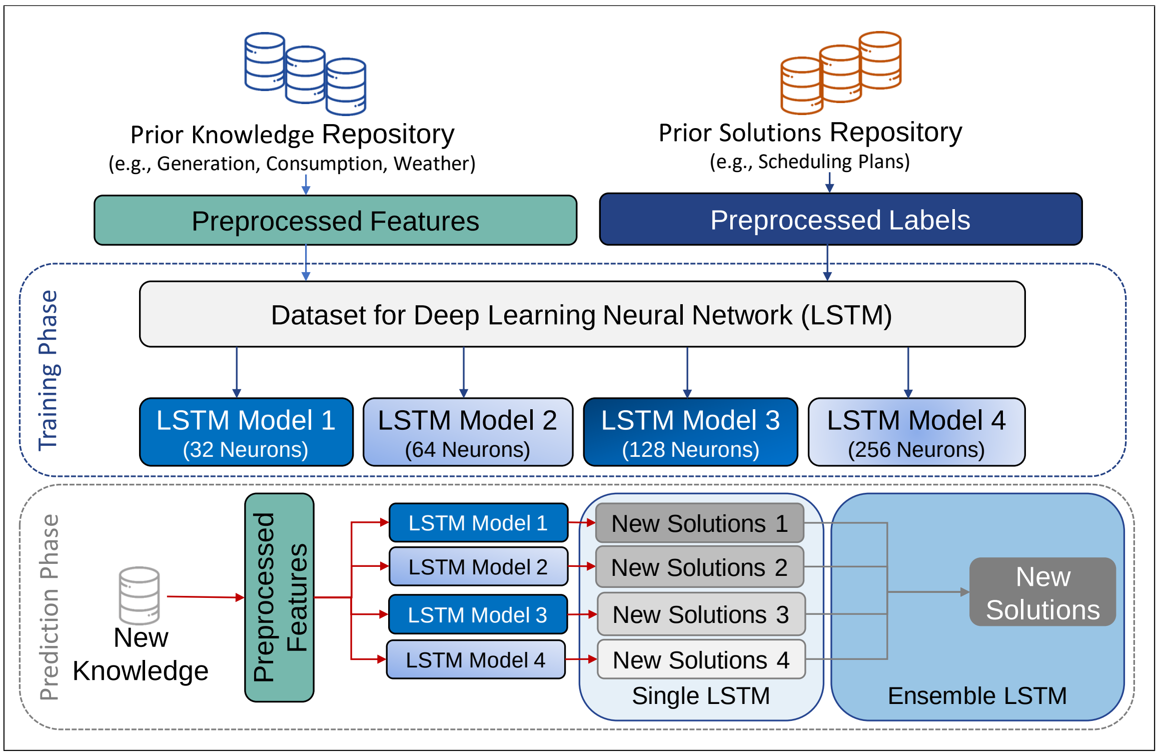


**Figure 6.13.:** Concept of supervised approach to assist EAs for scheduling DERs with its two phases, i.e., training and prediction phase

Section 5.2.1 –. Since the weather data is available only in a daily resolution, adding the same daily value of each weather factor to all input vectors for each time step introduces no significant effect. Therefore, the weather data is excluded from the input as a potential feature. The input is summarized by the encoder as one hidden state vector which is passed to the decoder as an initial hidden state. At each step of decoding, the decoder interprets and adapts the context of the state vector based on the corresponding label. Indeed, it predicts a vector for the dense layer which in turn predicts the output, i.e., a gene. Moreover, it updates the hidden state and passes it to the next time step. At the end of decoding, a sequence of genes that collectively represents a chromosome, i.e., a scheduling plan is generated.

Obviously, the labels, i.e., the chromosomes with their gene coding, represent a challenge for the LSTM model, since each gene has an Unit ID and three decision variables to be predicted. To tackle this problem, each variable of a gene is represented as a set of classes, namely 50 classes for the Unit ID, 24 classes for the Start, 24 classes for the Duration and 100 classes for PF – for simplicity's sake, PF is converted from a float to an integer representation with one percent precision –. This enables the dense layer to perform a one-hot-encoding by using a softmax activation function to predict the values of each variable of a gene. To this end, the decoder and dense layer are divided into multiple output layers, namely four layers where the four variables of a gene are synchronously predicted enabling the creation of the whole gene at each time step as shown in Figure 6.14. While the number of neurons in each layer of the dense layer is corresponding to the number of classes taken by each variable, i.e., 50, 24, 24 and 100, the number of neurons in the encoder and decoder

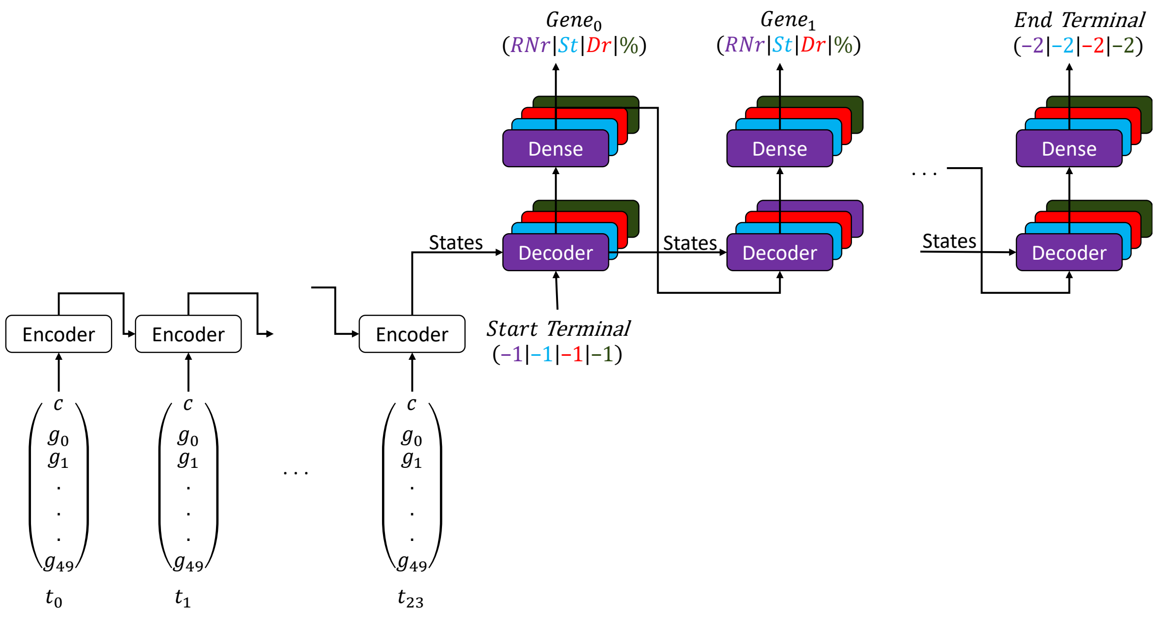
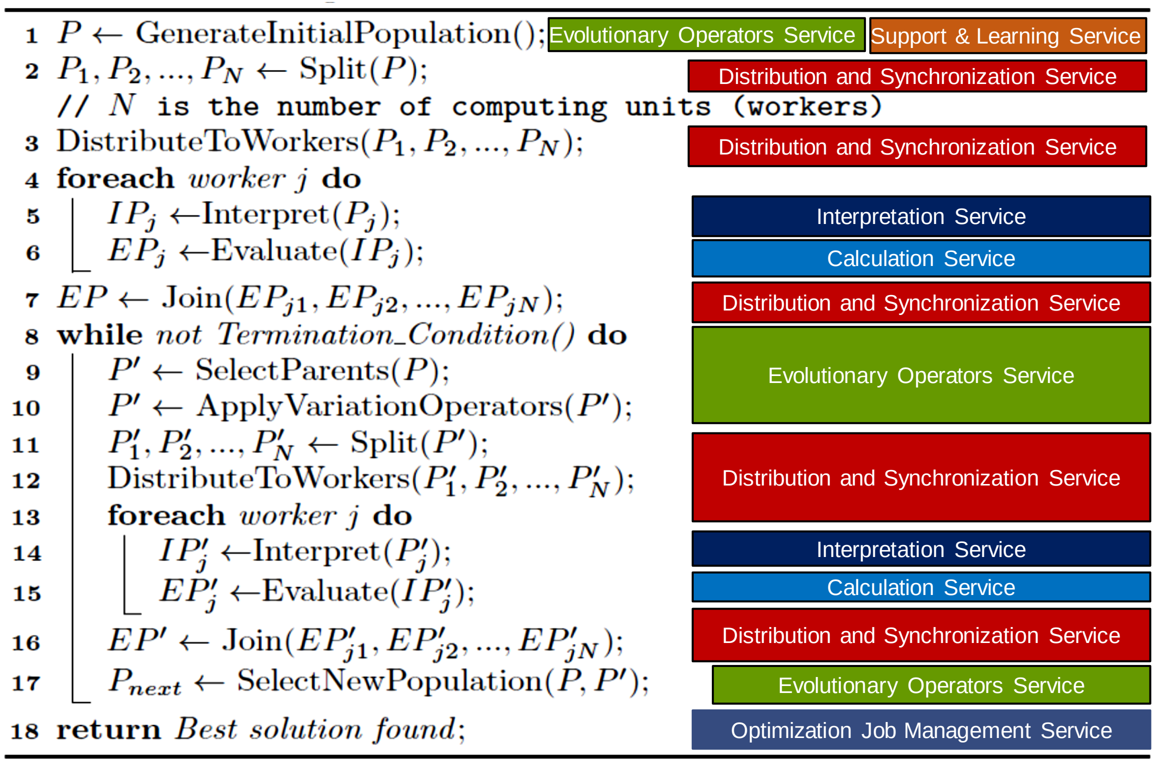


**Figure 6.14.:** Illustration of the Seq2Seq LSTM model architecture with multiple outputs

layers is independent of the representation of the output. Therefore, the proposed LSTM model supports different numbers of neurons for the encoder and decoder layers, namely 32, 64, 128 and 256 neurons. This allows the application of an ensemble method for creating a pool of solutions as shown in Figure 6.13. The proposed LSTM model only generates one scheduling plan per input representing a challenge when the requested number of similar solutions are more than one. Therefore, some noise about +/- 15% to the overall original input, i.e., consumption and generation are added to obtain multiple similar scheduling plans. The trend of consumption profile and the generation of each DER is preserved.

The S&L Service realizes the proposed ML hybridization approaches for assisting the parallelized EA GLEAM in solving the problem of scheduling DERs. The Global Model introduced in Section 2.2.1 and described in Section 4.2 is applied as a parallel model for EA for solving the problem of scheduling DERs. The main microservices of the Global Model, namely the Opt.J.M. Service, the Ds.S. Service, the E.O. Service and the Ca. Service are mapped to the pseudocode of the Global Model as shown in Figure 6.15. The E.O. Service executes lines 1, 8, 9, 10 and 17, the Ds.S. Service implements lines 2, 3, 7, 11, 12 and 16, the Ca. Service performs the evaluation by executing lines 6 and 15, the Opt.J.M. Service corresponds to line 18. For scheduling DERs, the additional service, namely the Interpretation Service explained in Section 5.2.3 is mapped to the pseudocode for executing lines 5 and 14. The S&L Service assists the E.O. service by performing the first line of the pseudocode.

Figure 6.16 illustrates the integration of the S&L Service into BeeNestOpt.IAI where it interacts only with the Opt.J.M. Service for seeding the initial population of the E.O. Service. The two main repositories, namely the ‘Prior Knowledge Repository’ and the ‘Prior Solutions Repository’ are stored in the persistent storage of the Data Layer of BeeNestOpt.IAI. To achieve that, PostgreSQL is used as a relational database to store both repositories.



```
$P \leftarrow$ GenerateInitialPopulation();
$P_1, P_2, ..., P_N \leftarrow$ Split($P$);
// $N$ is the number of computing units (workers)
DistributeToWorkers($P_1, P_2, ..., P_N$);
foreach $worker\ j$ do
    $IP_j \leftarrow$Interpret($P_j$);
    $EP_j \leftarrow$Evaluate($IP_j$);
$EP \leftarrow$ Join($EP_{j1}, EP_{j2}, ..., EP_{jN}$);
while $not\ Termination\_Condition()$ do
    $P' \leftarrow$ SelectParents($P$);
    $P' \leftarrow$ ApplyVariationOperators($P'$);
    $P'_1, P'_2, ..., P'_N \leftarrow$ Split($P'$);
    DistributeToWorkers($P'_1, P'_2, ..., P'_N$);
    foreach $worker\ j$ do
        $IP'_j \leftarrow$Interpret($P'_j$);
        $EP'_j \leftarrow$Evaluate($IP'_j$);
    $EP' \leftarrow$ Join($EP'_{j1}, EP'_{j2}, ..., EP'_{jN}$);
    $P_{next} \leftarrow$ SelectNewPopulation($P, P'$);
return $Best\ solution\ found$;
```

Evolutionary Operators Service
Support & Learning Service
Distribution and Synchronization Service
Distribution and Synchronization Service
Interpretation Service
Calculation Service
Distribution and Synchronization Service
Evolutionary Operators Service
Distribution and Synchronization Service
Interpretation Service
Calculation Service
Distribution and Synchronization Service
Evolutionary Operators Service
Optimization Job Management Service

**Figure 6.15.:** Mapping the proposed microservices of the Global Model with the Support-and-Learning-Service (brown hexagon) to the pseudocode of the Global Model of EA

Table 6.4 shows an example of the 'Prior Solutions Repository' where the chromosomes representing already found scheduling plans beside other information are stored. The 'id' column acts as the primary key of the table and is auto-incremented. The 'date' is the scheduled day, the 'plan_number' column is used to distinguish between multiple chromosomes, i.e., scheduling plans for the same date. The other columns, namely 'fitness', 'price', 'daily_deviation' and 'hourly_undersupply' are the values of fitness, objective functions and constraints related to this chromosome (see, Section 5.2.1).

| id | date | plan_ number | chromosome | fitness | price | daily_ deviation | hourly_ undersupply |
|---|---|---|---|---|---|---|---|
| 0 | 2012-04-03 | 0 | 3200 ... | 90247 | 92.04 | 0.01 | 0 |
| 1 | 2012-04-04 | 0 | 3200 ... | 70767 | 134.87 | 3.34 | 1 |
| ... | ... | ... | ... | ... | ... | ... | ... |

**Table 6.4.:** PostgreSQL database structure with some exemplary values for a potentially saved chromosome

### 6.3.3. Experimental Setup and Results

After testing the generality, modularity and flexibility of the S&L Service by integrating the above-explained ML hybridization approaches, namely the clustering-based and the LSTMs-based approaches, we assess their performance for enhancing the EA GLEAM in

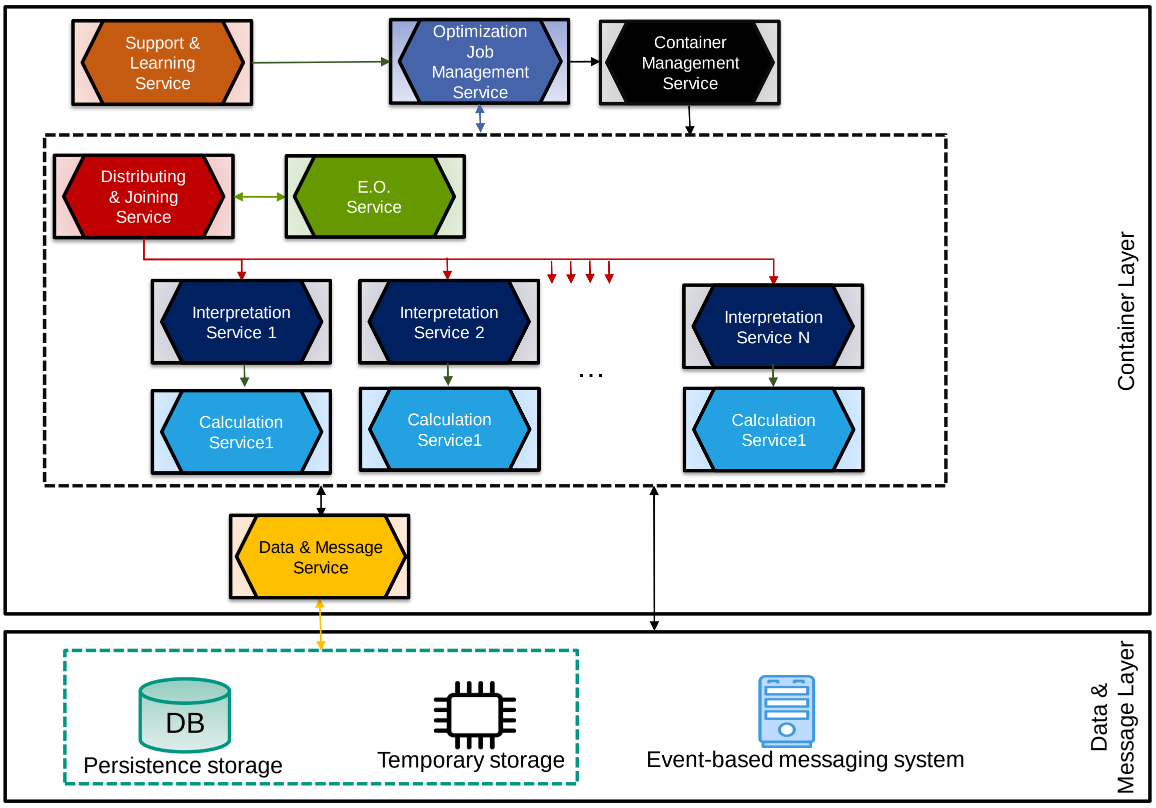
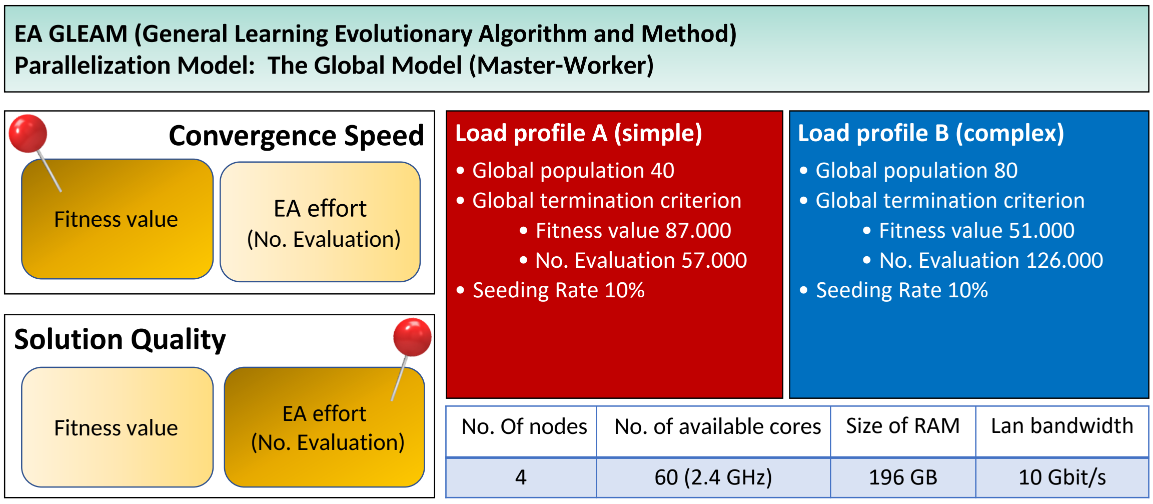


**Figure 6.16.:** Main microservices of the Global Model with Support-and-Learning-Service (brown hexagon) and Interpretation Service (dark blue hexagon) for scheduling DERs

terms of convergence speed and solution quality. While the convergence speed states how much effort the EA spends to reach a solution with a predefined quality, the solution quality metric reports which solution quality can be reached within a predefined EA effort. In other words for the first metric, a predefined fitness value is given as the termination criterion and the performance of the EA is measured based on the spent effort to reach such fitness. For the second metric, the EA effort is defined as the termination criterion while the fitness value is observed as shown in Figure 6.17.

For training the proposed ML models of the supervised and unsupervised hybridization approaches, the hourly day-ahead scheduling plans for one year, i.e., from 2010-07-01 to 2011-06-30 for the customer No. 102 are calculated to be used with the corresponding consumption, generation and weather data as a training dataset. For the parallelization model, 40 workers for the Global Model is used. The EA GLEAM is terminated after performing at minimum 200 generations and at maximum 500 generations according to the complexity of the load profile of the scheduled day. For testing the proposed ML hybridization approaches, the consumption profiles of the customer mentioned above are used. After applying Kmeans clustering algorithm, the consumption profiles show an unequal complexity, representing two groups of load profiles. While in the first group, the simple load profiles are grouped, in the second one, the complex profiles are gathered. From each group, one load profile is selected, i.e., a simple load profile and a complex one. As shown in Figure 6.18, the load profile for the date ‘2011-07-18’ (blue line) is more complex



**EA GLEAM (General Learning Evolutionary Algorithm and Method)**
**Parallelization Model: The Global Model (Master-Worker)**

**Convergence Speed**

Fitness value

EA effort (No. Evaluation)

**Solution Quality**

Fitness value

EA effort (No. Evaluation)

**Load profile A (simple)**
- Global population 40
- Global termination criterion
  - Fitness value 87.000
  - No. Evaluation 57.000
- Seeding Rate 10%

**Load profile B (complex)**
- Global population 80
- Global termination criterion
  - Fitness value 51.000
  - No. Evaluation 126.000
- Seeding Rate 10%

| No. Of nodes | No. of available cores | Size of RAM | Lan bandwidth |
|---|---|---|---|
| 4 | 60 (2.4 GHz) | 196 GB | 10 Gbit/s |

**Figure 6.17.:** Summary of the evaluation setup

than the other one (red line) for the date ‘2012-04-03’, since it has higher peaks and more sudden inclinations.

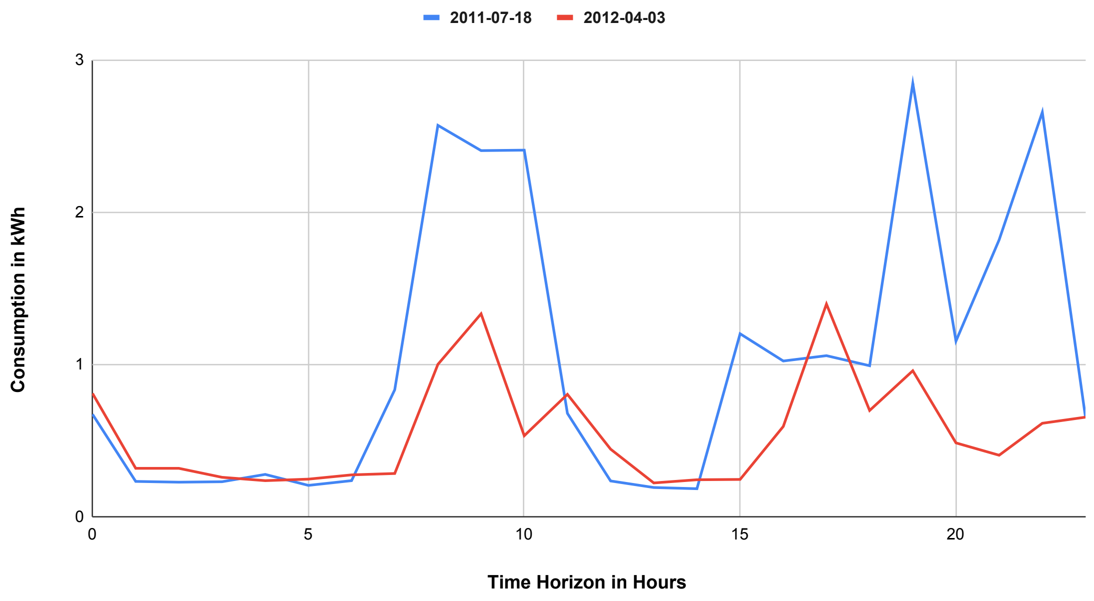


**Figure 6.18.:** Consumption profiles of customer No. 102 for the two considered use case scenarios

The stacked hourly generation for the 50 DERs on these days is illustrated in Figure 6.19 and 6.20.

In the context of the present evaluation, the EA effort is quantified by the number of fitness evaluations performed by the EA to find a solution with a given quality. For measuring the convergence speed, we experimentally fix the values of the fitness to be reached as follows: 87% and 51% of the highest fitness value which is set to ‘100.000’ for the simple load profile and the complex one, respectively. These two values are experimentally defined

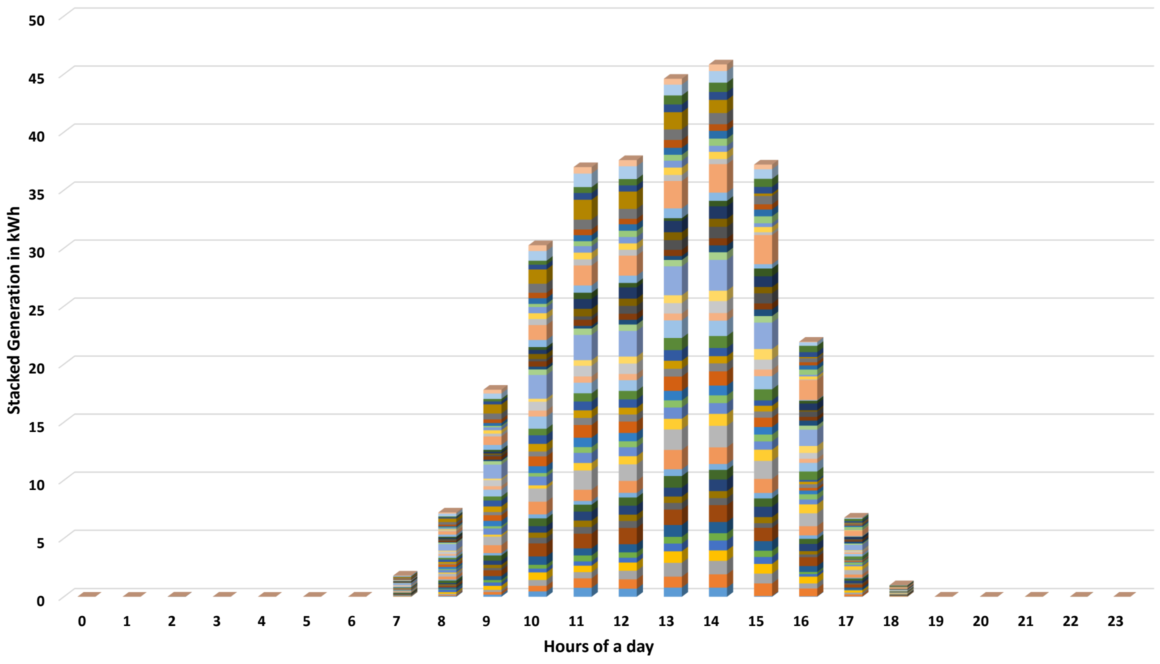


**Figure 6.19.:** Stacked generation of all 50 DERs on the 18. Juli 2011

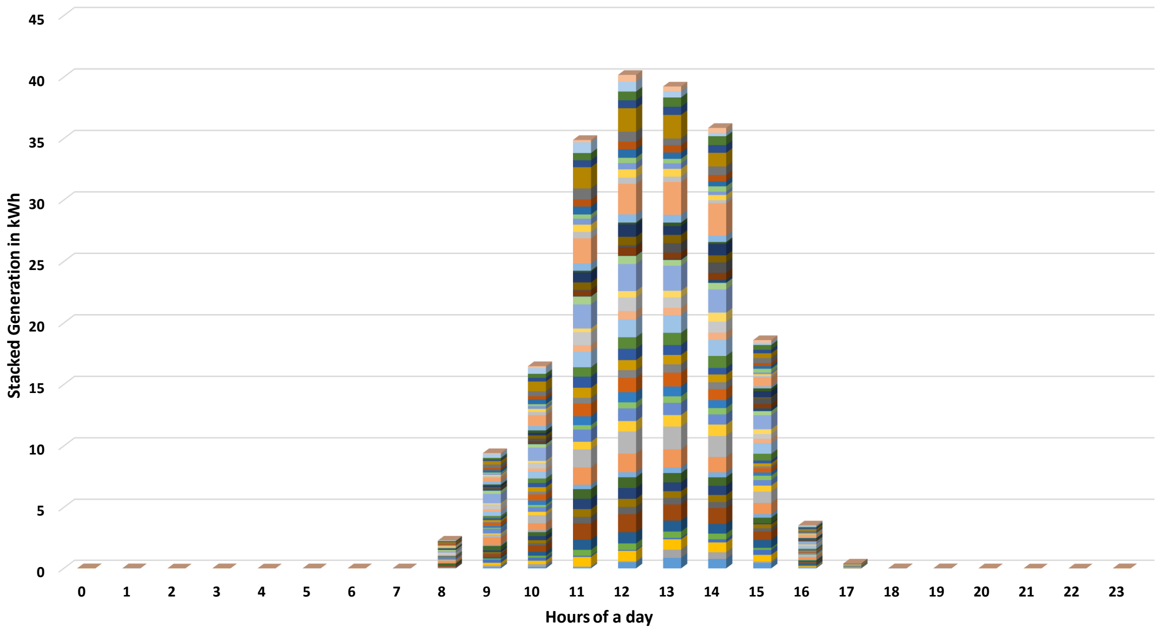


**Figure 6.20.:** Stacked generation of all 50 DERs on the 3. April 2012

and represent the average fitness values of 50 runs that can be reached by spending a moderate effort of the pure EA GLEAM, i.e., 200 generations. For the comparison of the solution quality, the EA GLEAM is terminated after performing 57.000 fitness evaluations for the simple load profile and 126.000 fitness evaluations for the complex one. Both values correspond with the previous fitness values, i.e., the EA GLEAM performs on average 57.000 and 126.000 fitness evaluations to get solutions with the aforementioned solution qualities. EAs have to be parameterized for generating solutions with high quality. There are various parameter settings to be tuned, e.g., population size, termination criterion, initial population policy and probability of applying genetic operators. These parameters have to be selected based on the complexity of the optimization problem, since they significantly change the behavior of the search procedure of EAs. For defining the appropriate population sizes to find good scheduling plans for the two considered load profiles, we perform multiple runs of the basic EA GLEAM, i.e., without using the hybridization approaches and then we take the average of them. We consider the amount of generations performed as a termination criterion. The obtained results for the simple load profile (A) show a clear trend toward smaller population sizes as shown in Table 6.5. By reducing the size of the population under 40 individuals, the EA GLEAM produces solutions with poor quality.

| Population | Generation | Fitness | Cost | Duration | Daily Deviation | Number of Hourly Deviation |
|---|---|---|---|---|---|---|
| 40 | 200 | 89500 | 90 | 468 | 4.8 | 0 |
| 80 | 120 | 86346 | 91 | 352 | 6.4 | 0 |
| 120 | 80 | 80375 | 102 | 301 | 8.8 | 0 |
| 200 | 40 | 73506 | 89 | 251 | 15 | 0 |

**Table 6.5.:** Population tuning with fixed amount of generations for 2012-04-03, i.e., the simple load profile (A)

Based on tuning results, two different population sizes depending on the complexity of the given load profile are selected, namely a population of 40 individuals for the simple load profile and a population of 80 individuals for the complex load profile – following the same concept of defining the population size of load profile (A) –. Moreover, the seeding rate of the initial population is set to 10% of the population size. The EA GLEAM with the selected settings and a completely random generated initial population is performed 10 times for each evaluation metric, i.e., convergence speed where the fitness value is set as the termination criterion and solution quality where the number of evaluations is set as the termination criterion. The results are averaged and serve as a reference value for assessing the performance of the hybridization approaches. The services are deployed on a cluster with four computing nodes where each node has 15 Intel cores (2,4 GHz), 196 GB RAM and an SSD disk. The nodes are connected to each other by a LAN with 10 GBit/s bandwidth (for more details about the deployment, see Section 4.4.2).

For evaluating the proposed ML hybridization approaches, the three modes, namely cluster, best and mix of the unsupervised hybridization approach and the LSTM model for the supervised hybridization approach with different feature combinations, i.e., c, g, w, cg, cw, gw and cgw are evaluated. The average of several runs, namely 10 times for each mode and the combination is taken to minimize the possible effect of the non-deterministic behavior of the clustering algorithm, EA and the LSTM model.

**Results** Tables 6.6 and 6.7 show our first concluded results for the convergence speed, i.e., the effort spent by the EA GLEAM to reach the predefined fitness value. Moreover,

| Mode | Feature Type | Simple Load Profile [2012-04-03] | Complex Load Profile [2011-07-18] | Simple Load Profile [2012-04-03] | Complex Load Profile [2011-07-18] |
|---|---|---|---|---|---|
| | | Number of Evaluations [average] | Number of Evaluations [average] | Fitness [average] | Fitness [average] |
| cluster | c | 43074.3 | 61707 | 86507.3 | 51132.5 |
| | cg | **41264.5** | 118179.8 | 88124.2 | **54912** |
| | cgw | 50376 | 112009.8 | 88023.9 | 50697.2 |
| | cw | 53492.1 | **57685.9** | 89759.6 | 49929.1 |
| | g | 43990.7 | 104229 | 89453.5 | 50657.8 |
| | gw | 44477.9 | 71880.2 | 89835.1 | 51961.7 |
| | w | 48744.1 | 71703.7 | **90481** | 48852 |
| best | c | 57614.8 | 92853.4 | 88274.7 | 48179.9 |
| | cg | 57423.3 | 115092.7 | 88970.2 | 50918 |
| | cgw | 45196.2 | 101741.6 | 88694 | 48869.4 |
| | cw | **43842** | 125075 | 88352.5 | 48761.2 |
| | g | 45669.4 | **88704** | 86054.4 | 48423.6 |
| | gw | 51626.5 | 133164.5 | **90407.9** | 49541.9 |
| | w | 52989.4 | 122673 | 87662.1 | **50925.5** |
| mix | c | 46666.9 | 97261.3 | 85673.8 | 50634 |
| | cg | 57110 | 83495 | **90651.3** | 51586.4 |
| | cgw | 59608.4 | 134459.7 | 89244.3 | 48268.6 |
| | cw | 49393.9 | 151148.4 | 90245.2 | **55689** |
| | g | 65141.7 | 122506.3 | 85126.5 | 51608.2 |
| | gw | 45019.9 | **52744** | 88418.1 | 49727.4 |
| | w | **41117** | 62922 | 86560.7 | 48597.6 |
| reference value | | 48202 | 100071 | 88570 | 50259 |

**Table 6.6.:** Results of the unsupervised approach. The best results, i.e., the lowest number of fitness evaluations and highest solution quality are highlighted per mode. The last row provides the results of pure EA GLEAM as reference value without the S&L Service

they present the first achieved results for the solution quality, i.e., the achieved fitness value within the predefined number of evaluations. The predefined fitness value is set to 87000 and 51000 for the simple load profile (2012-04-03) and the complex load profile (2012-07-18), respectively. For the solution quality, the predefined number of evaluations is set to 57000 and 126000 for the simple load profile (2012-04-03) and the complex load profile (2012-07-18), respectively. While Table 6.6 compares the reference results of pure EA GLEAM to the results obtained by applying the unsupervised hybridization approach to assist EA GLEAM, Table 6.7 compares the results of using the supervised hybridization approach, i.e., LSTM model to the reference values. On one hand, the results for the simple load profile shown in Figures 6.21 and 6.22 indicate an improvement in the convergence speed of the hybrid EA GLEAM for some features compared to the completely random

| **Mode** | **Feature Type** | **Simple Load Profile [2012-04-03]** | **Complex Load Profile [2011-07-18]** | **Simple Load Profile [2012-04-03]** | **Complex Load Profile [2011-07-18]** |
|---|---|---|---|---|---|
| | | Number of Evaluations [average] | Number of Evaluations [average] | Fitness [average] | Fitness [average] |
| LSTM | 32 | **45980.9** | **49699.7** | 89432.1 | 48940.3 |
| | 64 | 60216.2 | 100684.5 | 89289.9 | 51076.4 |
| | 128 | 48755.9 | 107953.8 | 87322.8 | 49185.2 |
| | 256 | 58411.7 | 98221.9 | **90047.9** | **51288.1** |
| | ensemble | 48202 | 97112.6 | 85803.4 | 48012.1 |
| reference value | | 48202 | 100071 | 88570 | 50259 |

**Table 6.7.:** Results of the supervised approach. The best results, i.e., the lowest number of fitness evaluations and highest solution quality are highlighted per mode. The last row provides the results of pure EA GLEAM as reference value without the S&L Service

seeding. Precisely, hybrid EA GLEAM requires less effort up to 15% to reach a solution with a fitness of 87.000. On the other hand, a less impact on the final solution quality is observed, where an enhancement of about 2% is measured as shown in Figures 6.23 and 6.24.

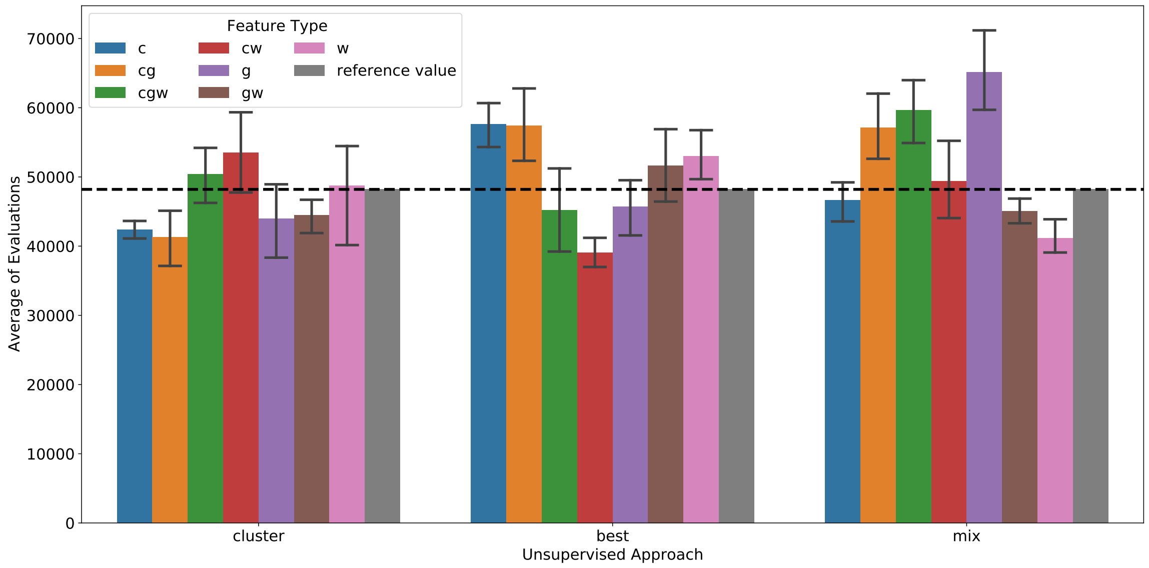


**Figure 6.21.:** Results of the convergence speed for the load profile A (simple) applying the unsupervised hybridization approach

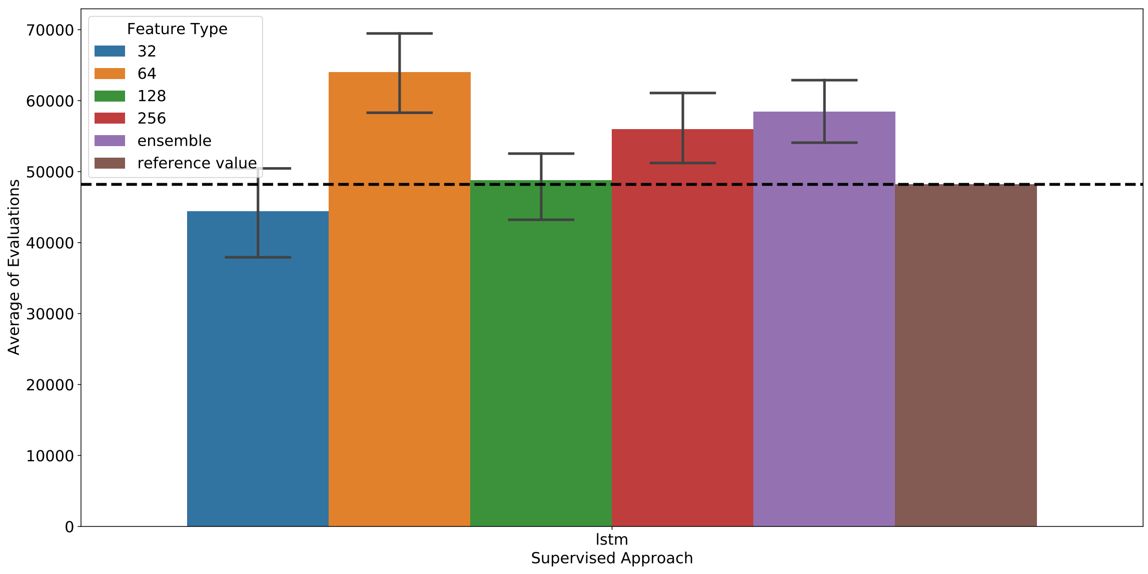


**Figure 6.22.:** Results of the convergence speed for the load profile A (simple) applying the supervised hybridization approach

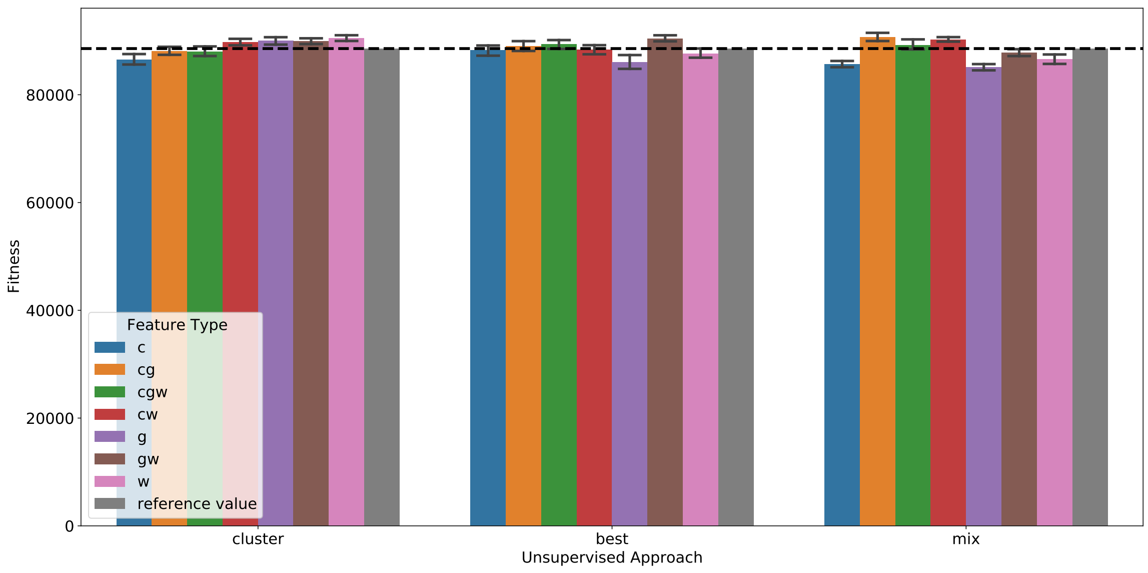


**Figure 6.23.:** Results of the solution quality for the load profile A (simple) applying the unsupervised hybridization approach

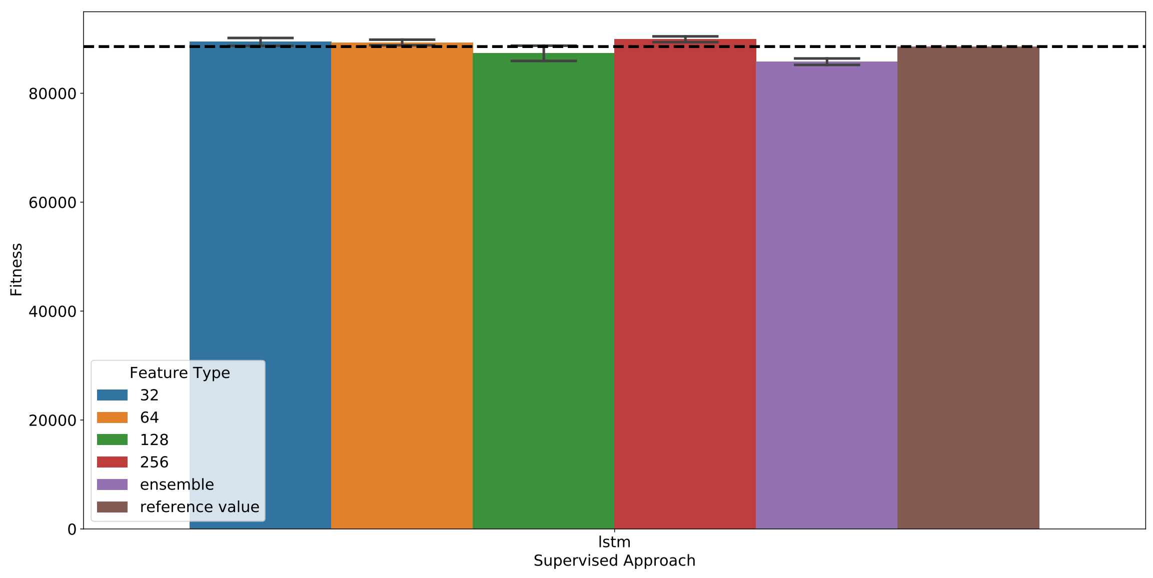


**Figure 6.24.:** Results of the solution quality for the load profile A (simple) applying the supervised hybridization approach

For the complex load profile, the obtained results show improvements in the convergence speed of the EA GLEAM using the proposed ML hybridization approaches compared to the completely random seeding. The hybridization approaches reduce the required effort up to 50% to reach a solution with a fitness of 51.000 as shown in Figures 6.25 and 6.26. However, a less impact on the final solution quality by applying the ML hybridization approaches is observed, where an enhancement of about 11% is measured as shown in Figures 6.27 and 6.28.

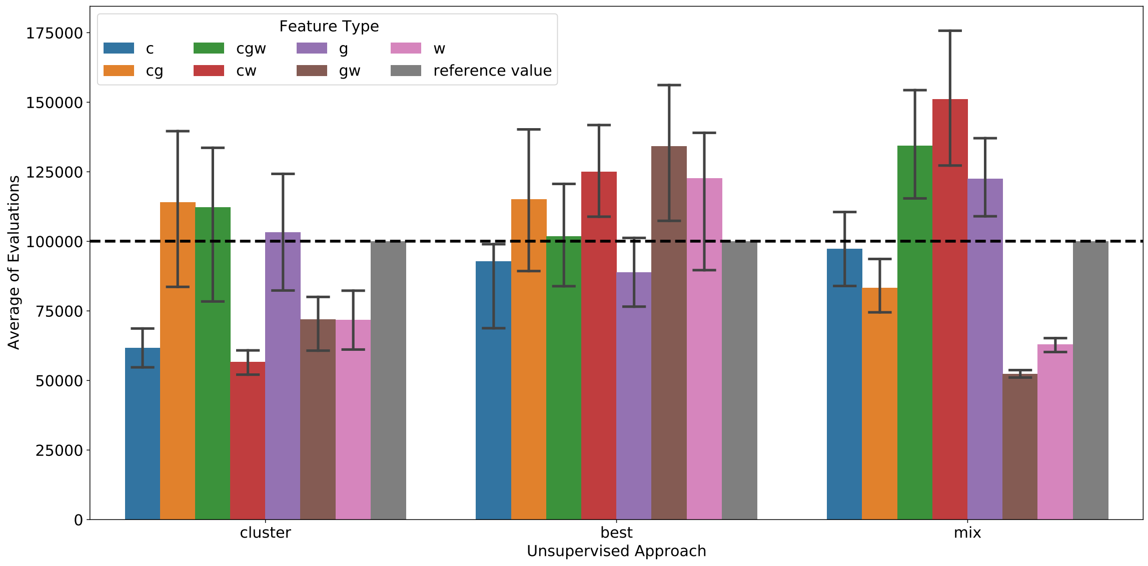


**Figure 6.25.:** Results of the convergence speed for the load profile B (complex) applying the unsupervised hybridization approach

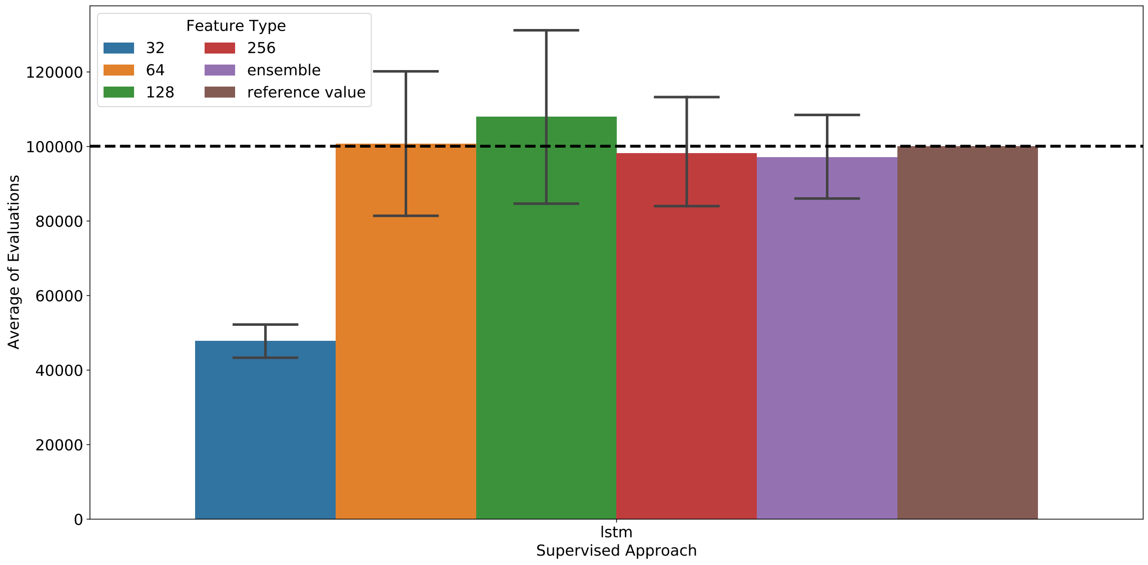


**Figure 6.26.:** Results of the convergence speed for the load profile B (complex) applying the supervised hybridization approach

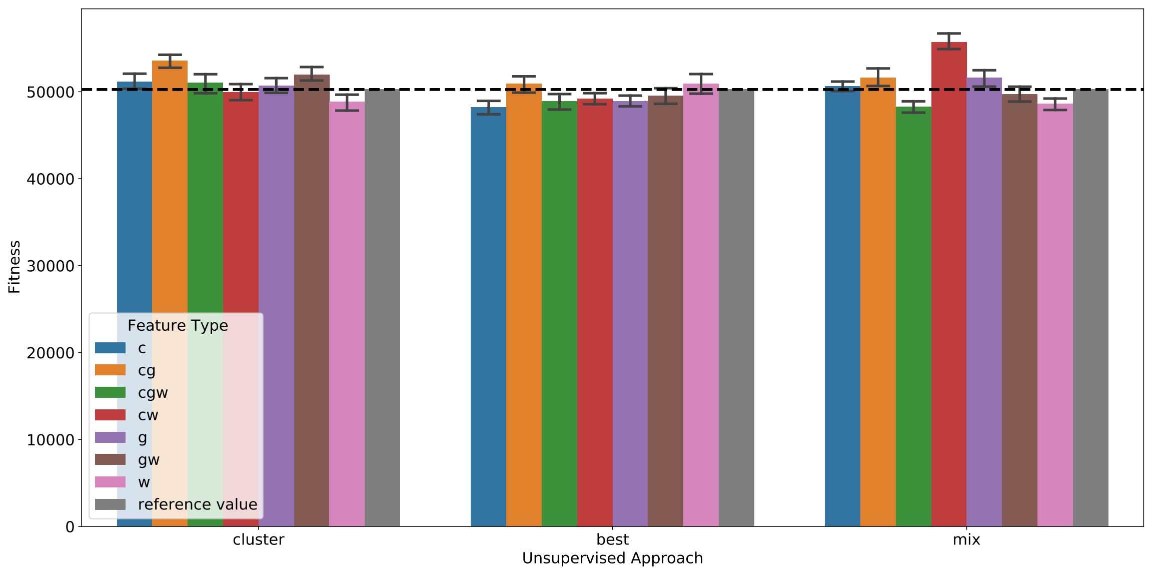


**Figure 6.27.:** Results of the solution quality for the load profile B (complex) applying the unsupervised hybridization approach

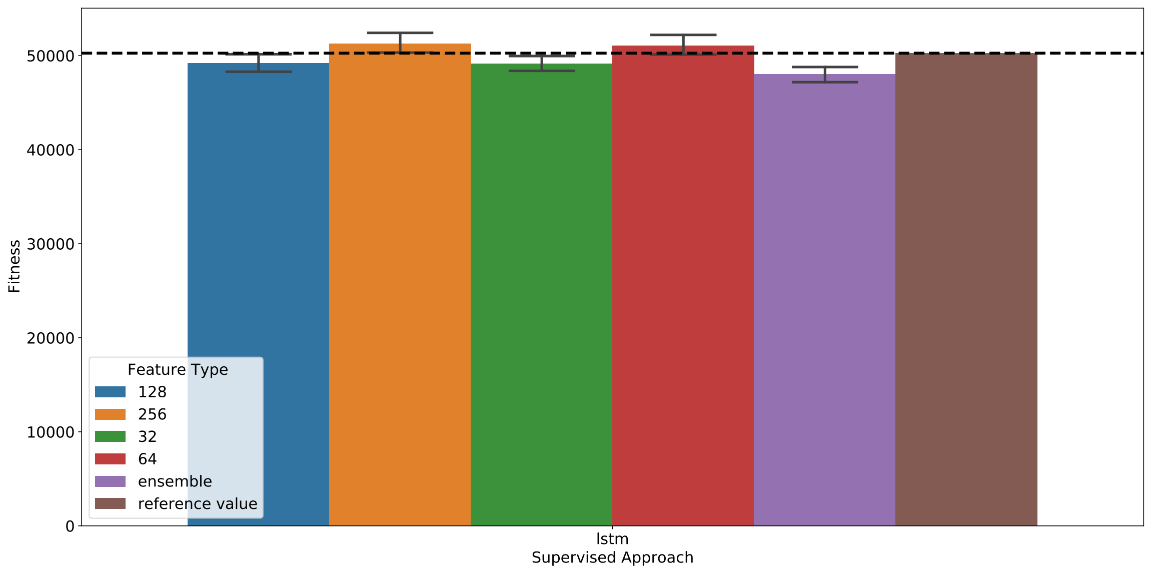


**Figure 6.28.:** Results of the solution quality for the load profile B (complex) applying the supervised hybridization approach

**Results Discussion** The results introduced in the above tables and figures show the averaged results of applying the proposed ML hybridization approaches over ten runs for each mode and feature combination. The best result of each mode with the corresponding feature is presented in Figure 6.29. The EA effort, i.e., the computational effort, is reduced up to 15% for the simple load profile by applying the mode mix with the weather features. Furthermore, an acceleration of about 50% for the complex load profile by using the LSTM model with 32 neurons per layer is achieved. It is noticeable that not all mode and feature combinations decrease the EA effort to reach a solution with the predefined fitness. For example, applying the mix mode with generation features increases such effort about 35% for the simple load profile and about 51% for the complex load profile with the consumption and weather features as shown in Table 6.6. This is due to the fact that some modes and features seed the initial population with inadequate solutions which disturb the search process of the EA. Consequently, more effort has to be spent by the EA to eliminate the effect of such disturbances. This is also the case for the solution quality metric where seeding the initial population does not obviously improve the fitness value compared to the reference value. For the fitness value of the simple load profile, there is no big enhancement. Indeed, only a small improvement up to 2% can be gained for the simple load profile as shown in Figure 6.29. However for the complex one, more enhancement up to 11% can be observed.

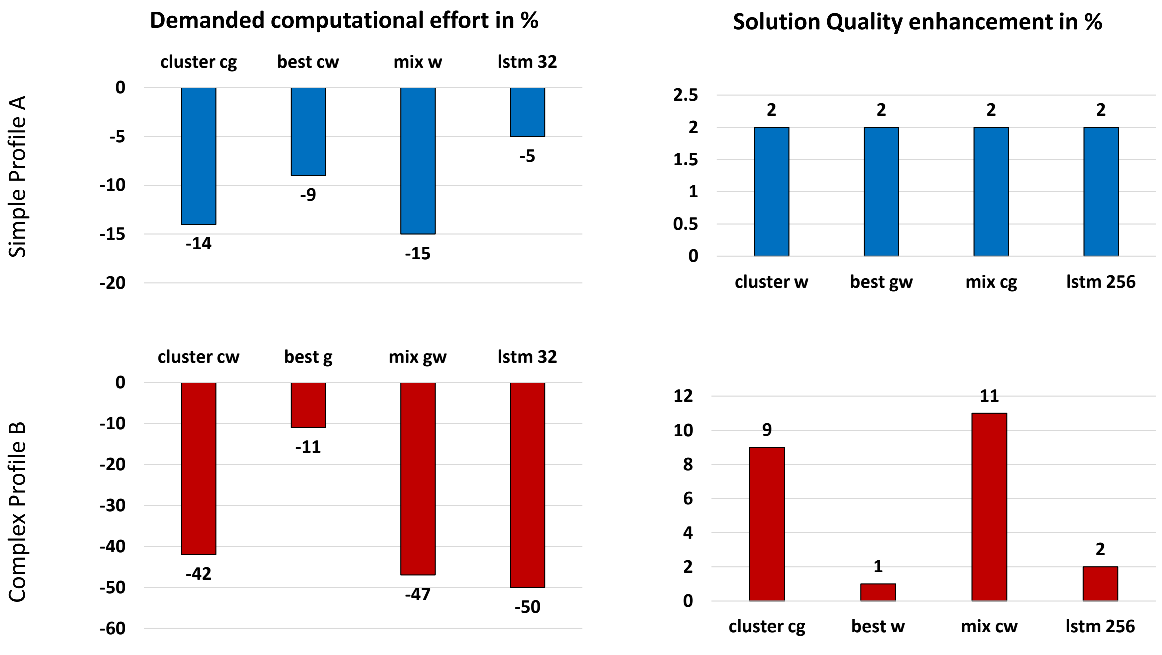


**Figure 6.29.:** Comparing the best results of both load profiles, A (simple) and B (complex)

To summarize, the obtained results in Tables 6.6 and 6.7 show that the seeding of the initial population with prior solutions or new solutions generated based on the knowledge learned from the prior ones can impact the behavior of the EA positively or negatively confirming the results of hybridizing EA introduced in the literature, e.g., [120, 175, 97, 63]. I.e, not every initial population seeding strategy has a positive effect on the convergence speed and final solution quality. Moreover, there is no one optimal mode with a specific feature combination that always has a positive effect on the search process of the EA for both considered load profiles. It is noticeable that applying the modes which use some randomness in the selection process of similar solutions, namely the cluster and mix modes outperform the exact mode, i.e., the best mode. This is due to the fact that such randomness increases the likelihood to seed the initial population with diverse solutions assisting the EA to discover new areas in the search space. The LSTM Model introduces no results to enhance the solution quality for both load profiles. The unconvincing performance of the LSTM model for improving the solution quality can be attributed to the lack of training data. The generated dataset has only 364 training samples which are insufficient to train such a complex model. However, for reducing the required computational effort, the LSTM shows at least one time a convincing performance by decreasing the required effort to reach the predefined fitness value for the complex load profile up to 50% compared to the reference value. This introduces LSTM as a promising approach to generate new solutions based on the gained knowledge from the prior ones for seeding the initial population of an EA. It is clear that the proposed ML hybridizing approaches for assisting the EA in solving the problem of scheduling DERs affect the convergence speed more than the solution quality. This is due to the fact that it is much harder to improve the fitness of the final solution introduced by the basic EA when this solution is the optimal one or very close to the optimal one. For example, the basic EA GLEAM reaches the optimal solution or the near-optimal one with a fitness of 87000 for the simple load profile. Nevertheless, an overall increase in convergence speed by utilizing the ML hybridizing approaches for both load profiles is observed. Consequently, seeding the initial population for solving the problem of scheduling DERs can accelerate the search process to solve the considered optimization problem. Moreover, it shows a potential in the enhancement of the solution quality.

## 6.4. Summary

In this chapter, a novel and generic microservice-based software method to facilitate the hybridization process of an EA in high-performance computing environments is introduced answering the research question **[RQ5]**. Beside microservices, two lightweight technologies, namely container virtualization and the publish/subscribe messaging paradigm are exploited to develop a modular, extendable, flexible, generic and scalable software solution to provide access to different hybridization methods. These three technologies together provide an easy hybridization of an existing EA, e.g., seeding of the initial population of EAs based on ML methods and models. Moreover, it guarantees a seamless deployment of several hybridization approaches in a scalable runtime environment and even in a big data environment paving the road for utilizing ML with EA for solving large-scale optimization problems. For evaluation purposes, BeeNestOpt.IAI is extended to integrate the proposed software solution. Its generality and applicability are tested by introducing two new hybridization approaches. The first hybridization approach, namely the unsupervised hybridization approach is realized as a clustering problem, in which the potential similar prior solutions are selected from the prior computed using clustering algorithms. The second hybridization approach, namely the supervised hybridization approach utilizes the Seq2Seq LSTM models with multiple-output layers to predict new solutions based on the gained knowledge from prior solutions. A use case scenario for solving the problem of scheduling DERs is defined for evaluating the applicability of the proposed ML hybridization approaches. This use case instruments 50 DERs and two load profiles, namely a simple load profile and a complex load profile for finding an optimal hourly day-ahead scheduling plan. In this evaluation, the EA GLEAM is exemplarily integrated into the microservice architecture to be assisted by both approaches for generating a part of the initial population. The Global Model as a parallelization model of EA is chosen. A high-performance computing environment, namely a cluster with 60 cores and 196 GB of RAM is used for testing the impact of both hybridization approaches on the behavior of the EA GLEAM. Such an impact is measured in terms of convergence speed and solution quality. The first observed results differ depending on the complexity of the optimization task. For the simple load profile, a little solution quality enhancement of up to 2% is obtained. However, a speedup in the convergence speed is measured with up to 15% reduction of the computational effort to reach a predefined fitness. The results of the complex load profile show more performance enhancements, i.e., an improvement of up to 11% in the solution quality and a reduction of 50% in the convergence speed. These promising results show the potential of the proposed hybridization approaches to accelerate EAs for solving complex optimization problems.

# 7. Summary and Outlook

## 7.1. Summary

Population-based metaheuristics like Evolutionary Algorithms (EAs) are general optimization methods that are applied to solve a plethora of applications in business, commerce and engineering, to name a few. Their generality arises from the fact that they can solve any optimization problem with little problem-specific knowledge. Moreover, EAs are good alternatives for finding solutions with good quality for complex and large-scale optimization problems whereas other optimization techniques, namely the exact and heuristics methods provide solutions with poor quality or require too much time [3]. It is obvious that the exploration process of EAs to find a feasible solution in a complex and large-scale search space is significantly time-consuming, requires high computational demands. This occurs particularly in modern applications where the search space becomes more complex due to, e.g., multi domain-simulation. This limits the application of EAs on modern real-world optimization problems. In this thesis, our contributions for facilitating the usage and increasing the applicability of EAs in cluster computing environments based on modern software and hardware are presented. The first contribution focuses on conceptualizing and developing a generic and modular software architecture which is called BeeNestOpt.IAI based on microservices, container virtualization and the publish/subscribe messaging paradigm. BeeNestOpt.IAI facilitates the usage of EAs in cluster computing environments such as a cluster or a cloud answering three research questions **[RQ1, RQ3 and RQ4]**. This architecture is a combination of modular components known as microservices where each one can be scaled and developed independently. Each microservice is deployed and executed in one container or more. A set of APIs are designed to achieve communication among the microservice, the integration of any existing EA and the communication to other tools and services like simulators written in different programming languages. Moreover, BeeNestOpt.IAI provides a runtime environment for accelerating EAs by parallelizing their execution in cluster computing and cloud runtime infrastructures. It is designed to support the realization of any parallelization model from the three basic models, namely the Global Model (Master-Worker Model), the Coarse-Grained Model (Island Model) and the Fine-Grained Model. In the proposed software architecture, we distinguish between the functionalities related to the parallelization model and the ones related to the framework such as coordination. Firstly, the Global Model is realized and evaluated by developing five different microservices that carry out the functionalities of this model. Secondly, the Coarse-Grained Model is mapped to the proposed software architecture by adding new microservices or adapting the existing ones. Indeed, six decoupled and cohesive microservices carry out the tasks related to the Coarse-Grained Model. Both models and the functionalities of the proposed software solution are evaluated by using an existing EA, namely GLEAM where the entire framework is deployed on a cluster with 4 nodes and 128 cores for performance measurements. The scalability of the proposed software architecture in terms of the communication overhead and the parallel performance, i.e., speedup is presented and discussed. For the Global Model, the scalability is measured by varying the number of workers between 1 and 120 workers with a delay time for evaluating each individual set to 0.9 seconds. The network load, i.e., communication overhead is observed by changing the population size between 120 and 960 individuals. It is worth noting that the communication overhead is low and only slightly increasing in proportion to the size of the population. By analyzing the obtained results, we can conclude that the Global Model based on microservices and container technology is useful for an optimization problem with an evaluation time of more than 0.1 seconds per individual. The speedup test shows that the Global Model can achieve an almost linear speedup with an increase in the number of workers. Indeed, BeeNestOpt.IAI is able to guarantee an increase in the speedup up to a specific threshold which is reached by using a number of workers ranging between 50 and 60. For the Coarse-Grained Model, the overhead and the speedup are evaluated and discussed. For assessing the migration overhead, four communication topologies, namely Ring, Bi-Ring, Ladder and Complete with several configurations setup such as varying the migration rate between 1 and 16384 individuals and the number of islands between 1 and 120 are applied. Moreover, for simulating different complexities of optimization problems, the delay feature of the applied GLEAM version is used and set to different evaluation times ranging between 1 ms and 32 ms. The obtained results show that the underlying microservice architecture for realizing the Coarse-Grained Model introduces an almost constant migration overhead for a migration rate lower than 128 migrants and substantial speedup rates.

The second contribution of the present thesis builds upon the first one by focusing on enhancing the parallel performance of the three basic parallelization models answering three research questions **[RQ2, RQ3 and RQ4]**. In this contribution, we introduce our software architecture to hierarchically combine arbitrary parallelization models from the three basic ones with minimum adaptation efforts. We demonstrate the flexibility, generality and modularity of BeeNestOpt.IAI by combining the Coarse-Grained Model with the Global Model using eight microservices. This hybrid model eliminates the drawbacks of each model and simultaneously combines their advantages. A cluster computer is used to deploy the proposed microservices to form the Coarse-Grained - Global Hybrid Model. The problem of scheduling distributed energy resources is selected as a real-world optimization problem. This enables us not only to evaluate the performance of the proposed software architecture in terms of scalability and communication overhead but to also assess the ability of parallel EAs to solve such a scheduling problem. The evaluation is performed in three steps. In the first step, we evaluate the Global Model to solve this problem by changing the number of workers between 5 and 30 workers, whereas in the second step, we evaluate the Coarse-Grained Model by comparing four communication topologies, namely Ring, Bi-Ring, Ladder and Complete and varying the number of islands between 5 and 20 islands. Moreover, a synchronous migration policy with homogenous EA configuration are compared with an asynchronous migration policy and heterogeneous EA configuration. In the last step, the scalability and communication overhead of the Coarse-Grained - Global

Hybrid Model is evaluated by changing the number of islands and workers. It is noticeable that the proposed software solution facilitates the process of combining two parallelization models into one hierarchical model by increasing the parallel performance and at the same time allowing the communication overhead to be on reasonable levels. Precisely, the Coarse-Grained - Global Hybrid Model guarantees a speedup up to 4 times compared to its basic models and up to 75 times compared to the sequential execution of the EA. Moreover, by applying the heterogeneous configuration and the asynchronous migration an additional speedup is achieved. The third contribution of this thesis focuses on expanding the problem classes of EAs by extending BeeNestOpt.IAI to facilitate the hybridization process of any existing EAs with other algorithmic approaches in cluster computing environments which hence answers the fifth research question **[RQ5]**. The proposed solution allows the hybridization of the step of generating the initial population of existing EAs with any algorithms such as machine learning algorithms. This opens a new perspective to assist EAs by solving a wide range of optimization problems. In order to analyze the feasibility of the proposed design, two machine learning-based hybridization approaches are proposed. While in the first approach, an unsupervised technique, namely clustering is realized, in the second approach, a supervised technique, i.e., Sequence to Sequence (Seq2Seq) Long Short-Term Memory (LSTM) models with multiple-output-layers is used. Both approaches have the same task of assisting EAs by generating a part of the initial population based on domain-based knowledge. However, they differ in the technique used to seed the initial population. In the unsupervised approach, some historical solutions are selected based on external knowledge. In the supervised approach, new solutions are generated based on the learned knowledge gained from the historical solutions. We combine the hybridization approaches of EA with the parallel EA to solve the problem of scheduling distributed energy resources. In such a way, we evaluate BeeNestOpt.IAI and the two corresponding developed hybridization approaches. For measuring the convergence speed and the solution quality, two use case scenarios are defined. These use case scenarios are distinguished in their complexity. While a simple load profile is used in the first use case, a more complex one is considered in the second one. The results show that assisting the EA by seeding the initial population does not introduce a sufficient impact on the solution quality of the first use case scenario. However, a good speedup in the convergence speed is gained. For the second use case scenario, both proposed approaches enhance the solution quality and accelerate the search process.

To summarize and as shown in Table 7.1, the contributions of this thesis fulfill the main gaps of the mentioned libraries and frameworks reviewed in Section 3. Precisely, BeeNestOpt.IAI supports the parallelization of any existing EAs according to any parallelization model of the three basic ones. Moreover, it provides a simple and applicable mechanism to enhance the parallel performance of the basic models by combining them in a hierarchical structure with minimal adaptation efforts. One main feature that distinguishes our software architecture from others is that it allows the hybridization of any part of a parallel EA with any algorithmic approach. Besides these three beneficial traits, we facilitate the deployment of any existing EA in cluster computing environments or a cloud, since it is designed as a web-based application providing an easy-to-use management and execution environment.

| Framework or library | Global | Coarse-Grained | Fine-Grained | Hierar-chical | Pool | Hybrid-ization | Supporting cloud or cluster | Integration of existing EAs | Cooperating with external simulators | Hiding the technical aspects |
|---|---|---|---|---|---|---|---|---|---|---|
| **MALLBA<br>[8, 7]** | - | + | - | - | - | + | - | - | - | - |
| **ParadisEO<br>[33]** | + | + | - | + | - | + | - | - | - | - |
| **DREAM<br>[145]** | - | + | - | - | - | - | - | - | - | - |
| **DEAP<br>[62]** | + | + | - | - | - | - | - | - | - | - |
| **JCLEC<br>[181]** | + | - | - | - | - | + | - | - | - | - |
| **MRPGA<br>[91]** | + | - | - | - | - | - | + | - | - | - |
| **Verma et al.<br>[183]** | + | - | - | - | - | - | + | - | - | - |
| **elephant56<br>[165]** | + | + | + | - | - | - | + | - | - | - |
| **Martino et al.<br>[45]** | + | + | + | - | - | - | + | - | - | - |
| **Fazenda et al.<br>[52]** | - | + | - | - | - | - | + | - | - | - |
| **Flex-GP<br>[170]** | - | + | - | - | - | - | + | - | - | - |
| **jMetalSP<br>[136]** | + | + | + | - | - | - | + | - | - | - |
| **AMQPGA<br>[163, 164]** | + | - | - | - | - | - | + | - | - | - |
| **KafkEO<br>[73]** | - | - | - | - | + | - | + | - | - | - |
| **Lim et al.<br>[115]** | - | - | - | + | - | - | + | - | - | - |
| **P-CAGE<br>[61]** | - | - | - | + | - | - | + | - | - | - |
| **evospace-js<br>[66]** | - | - | - | - | + | - | + | - | - | - |
| **Meri et al.<br>[131]** | - | - | - | - | + | - | + | - | - | - |
| **Roy et al.<br>[161]** | - | - | - | - | + | - | + | - | - | - |
| **Merelo et al.<br>[130]** | - | - | - | - | + | - | + | - | - | - |
| **Jurczuk et al.<br>[93, 94, 95]** | + | - | - | - | - | - | + | - | - | - |
| **Luong et al.<br>[121]** | - | + | - | - | - | - | + | - | - | - |
| **BeeNestOpt.IAI** | + | + | + | + | - | + | + | + | + | + |

**Table 7.1.:** Comparison of the existing frameworks for parallel EAs to the BeeNestOpt.IAI

Furthermore, our proposed software architecture achieves seamless cooperation between EAs and other external tools and services like simulators.

## 7.2. Outlook

The proposed software architecture – BeeNestOpt.IAI – to increase the usage of EAs shows that using modern software technologies on powerful hardware opens new perspectives to face the main drawbacks of EAs such as scalability. Furthermore, it allows the extension of the application space of EAs to include more complex and large-scale optimization problems coming from Big Data applications, where large amounts of data are generated by sensing technologies and other Internet of Things (IoT) applications. The present thesis answers the four research questions introduced in Section 1.2 and paves the road for further research to enhance EAs, e.g., by reducing the amount of execution time and achieving better solution quality.

In the first contribution of this thesis which answers the following three research questions:

- **Research Question 1 [RQ1]:** Which software design is adequate for parallelizing EAs according to the basic parallelization models in a cluster computing environment?
- **Research Question 3 [RQ3]:** What are the adequate methods to facilitate the usage of existing EAs in a cluster computing environment?
- **Research Question 4 [RQ4]:** What are the adequate methods to facilitate the interaction between EAs and external tools and applications (e.g., simulators)?

We only focus on the three basic parallelization models, namely the Global Model (Master-Worker Model), the Fine-Grained Model (grid model) and the Coarse-Grained Model (Island Model). However, it is worthwhile to test if the proposed software architecture is able to realize other models such as non-classical parallelization models for EAs, e.g., the Pool Model. For the Coarse-Grained Model (Island Model), four static communication technologies are studied in terms of speedup and communication overhead. Nevertheless, it is recommended to extend the present software architecture to dynamically change the topologies during one optimization task building dynamic topology.

The second contribution of this thesis answers the second research question, namely

- **Research Question 2 [RQ2]:** Which benefits can be obtained from combining two or more parallel models of EAs into one parallel solution in a cluster computing environment?

For increasing the parallel performance of the basic parallelization models and for intensively evaluating the flexibility, generality and modularity of the BeeNestOpt.IAI, other hybridizations of the parallelization models for EAs, e.g., the Coarse - Fine-Grained hybrid Model or the Coarse - Coarse-Grained hybrid Model have to be comprehended and analyzed.

For increasing the performance and the applicability of EAs, this thesis introduces a new software method to hybridize parallel EA in cluster computing environment and proposes two approaches to hybridize EA with ML algorithms to seed the initial population considering domain-based knowledge answering the fifth research question, i.e.,

- **Research Question 5 [RQ5]:** Which software design is adequate for hybridizing of existing EAs with other algorithms in a cluster computing environment?

However, the same seeding or an adopted approach can be applied to other elements of an EA, e.g., the parent or survivors could be partially seeded by the proposed ML hybridization approaches or by adding adequate local search heuristics to improve descendants or to generate offspring. Additionally, other hybridization approaches, e.g., using neural networks for evaluating the proposed solutions can be applied and studied. Finally, it is of utmost importance to integrate the whole proposed software into a complex workflow that consists of different algorithms, software components, tools, services and simulators in order to solve more complex and realistic optimization problem such as the Energy Management System (EMS).

# Bibliography

[1] Ravindra K Ahuja, James B Orlin, and Ashish Tiwari. “A greedy genetic algorithm for the quadratic assignment problem”. In: *Computers&Operations Research*, Vol. 27, No. 10 (2000), pp. 917–934. DOI: `10.1016/S0305-0548(99)00067-2` (cit. on pp. 4, 44).

[2] Enrique Alba. “Parallel evolutionary algorithms can achieve super-linear performance”. In: *Information Processing Letters*, Vol. 82, No. 1 (2002), pp. 7–13. DOI: `10.1016/S0020-0190(01)00281-2` (cit. on pp. 3, 78, 79, 93, 126).

[3] Enrique Alba. *Parallel metaheuristics: a new class of algorithms*. Vol. 47. (2005). DOI: `10.1002/0471739383` (cit. on pp. 1, 167).

[4] Enrique Alba and Marco Tomassini. “Parallelism and evolutionary algorithms”. In: *IEEE transactions on evolutionary computation*, Vol. 6, No. 5 (2002), pp. 443–462. DOI: `10.1109/TEVC.2002.800880` (cit. on pp. 17, 18, 21).

[5] Enrique Alba, José M Troya, et al. “A survey of parallel distributed genetic algorithms”. In: *Complexity*, Vol. 4, No. 4 (1999), pp. 31–52. DOI: `10.5555/315491.315495` (cit. on pp. 18, 21).

[6] Enrique Alba and José M Troya. “Improving flexibility and efficiency by adding parallelism to genetic algorithms”. In: *Statistics and Computing*, Vol. 12, No. 2 (2002), pp. 91–114. DOI: `10.1023/A:1014803900897` (cit. on p. 21).

[7] Enrique Alba et al. “Efficient parallel LAN/WAN algorithms for optimization. The MALLBA project”. In: *Parallel Computing*, Vol. 32, No. 5–6 (2006), pp. 415–440. DOI: `10.1016/j.parco.2006.06.007` (cit. on pp. 37, 47, 170).

[8] Enrique Alba et al. “MALLBA: A library of skeletons for combinatorial optimisation”. In: *European Conference on Parallel Processing*. (2002), pp. 927–932. DOI: `10.1007/3-540-45706-2_132` (cit. on pp. 3, 37, 47, 170).

[9] Florian Allerding et al. “Electrical load management in smart homes using evolutionary algorithms”. In: *European Conference on Evolutionary Computation in Combinatorial Optimization*. (2012), pp. 99–110. DOI: `10.1007/978-3-642-29124-1_9` (cit. on p. 110).

[10] George S Almasi and Allan Gottlieb. *Highly parallel computing*. (1994) (cit. on p. 27).

[11] Lázaro Alvarado-Barrios et al. “An evolutionary computational approach for the problem of unit commitment and economic dispatch in microgrids under several operation modes”. In: *Energies*, Vol. 12, No. 11 (2019), p. 2143. DOI: `10.3390/en12112143` (cit. on pp. 108, 110).

[12] Alex M Andrew. “Introduction to evolutionary computing”. In: *Kybernetes.* (2004). DOI: 10.1108/03684920410699216 (cit. on p. 14).

[13] M Arenas et al. “JEO: a framework for Evolving Objects in Java”. In: *Actas Jornadas de Paralelismo*, Vol. 1 (2001) (cit. on p. 38).

[14] Maribel García Arenas et al. “A framework for distributed evolutionary algorithms”. In: *International Conference on Parallel Problem Solving from Nature.* (2002), pp. 665–675. DOI: 10.1007/3-540-45712-7_64 (cit. on p. 3).

[15] Ehsan Asadi et al. “Multi-objective optimization for building retrofit: A model using genetic algorithm and artificial neural network and an application”. In: *Energy and Buildings*, Vol. 81 (2014), pp. 444–456. DOI: 10.1016/j.enbuild.2014.06.009 (cit. on pp. 23, 45).

[16] Alireza Askarzadeh. “A memory-based genetic algorithm for optimization of power generation in a microgrid”. In: *IEEE transactions on sustainable energy*, Vol. 9, No. 3 (2017), pp. 1081–1089. DOI: 10.1109/TSTE.2017.2765483 (cit. on pp. 4, 46, 110).

[17] Masri Ayob and Graham Kendall. “A survey of surface mount device placement machine optimisation: Machine classification”. In: *European Journal of Operational Research*, Vol. 186, No. 3 (2008), pp. 893–914. DOI: 10.1016/j.ejor.2007.03.042 (cit. on p. 1).

[18] Thomas Bäck, Frank Hoffmeister, and Hans-Paul Schwefel. “A survey of evolution strategies”. In: *Proceedings of the fourth international conference on genetic algorithms.* (1991) (cit. on p. 11).

[19] Muhammed Fatih Balin, Abubakar Abid, and James Zou. “Concrete autoencoders: Differentiable feature selection and reconstruction”. In: *International conference on machine learning.* (2019), pp. 444–453 (cit. on p. 31).

[20] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE transactions on neural networks*, Vol. 5, No. 2 (1994), pp. 157–166. DOI: 10.1109/72.279181 (cit. on p. 33).

[21] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE transactions on neural networks*, Vol. 5, No. 2 (1994), pp. 157–166. DOI: 10.1109/72.279181 (cit. on p. 136).

[22] Jörg Biethahn and Volker Nissen. *Evolutionary algorithms in management applications.* (2012). DOI: 10.1007/978-3-642-61217-6 (cit. on p. 1).

[23] Christian Blume and Wilfried Jakob. *GLEAM - General Learning Evolutionary Algorithm and Method: Ein evolutionärer Algorithmus und seine Anwendungen.* Vol. 32. (2009). DOI: 10.5445/KSP/1000013553 (cit. on pp. 24–26, 79, 81).

[24] Christian Blume and Wilfried Jakob. “GLEAM-An Evolutionary Algorithm for Planning and Control Based on Evolution Strategy”. In: *GECCO Late Breaking Papers.* (2002), pp. 31–38 (cit. on pp. 24–26, 45, 79, 118).

[25] Christian Blume and Andrea Roli. “Metaheuristics in combinatorial optimization: Overview and conceptual comparison”. In: *ACM computing surveys (CSUR)*, Vol. 35, No. 3 (2003), pp. 268–308. DOI: `10.1145/937503.937505` (cit. on p. 1).

[26] Marlon Alexander Braun et al. “A neuro-genetic approach for modeling and optimizing a complex cogeneration process”. In: *Applied Soft Computing*, Vol. 48 (2016), pp. 347–358. DOI: `10.1016/j.asoc.2016.07.026` (cit. on p. 13).

[27] Alexander EI Brownlee et al. “Using a Markov network as a surrogate fitness function in a genetic algorithm”. In: *IEEE Congress on Evolutionary Computation*. (2010), pp. 1–8. DOI: `10.1109/CEC.2010.5586548` (cit. on p. 45).

[28] Peter Brucker, Yu N Sotskov, and Frank Werner. “Complexity of shop-scheduling problems with fixed number of jobs: a survey”. In: *Mathematical Methods of Operations Research*, Vol. 65, No. 3 (2007), pp. 461–481. DOI: `10.1007/s00186-006-0127-8` (cit. on p. 110).

[29] Eduard Bullich-Massaguí et al. “Microgrid clustering architectures”. In: *Applied energy*, Vol. 212 (2018), pp. 340–361. DOI: `10.1016/j.apenergy.2017.12.048` (cit. on p. 108).

[30] Tadeusz Burczyński et al. “Optimization and defect identification using distributed evolutionary algorithms”. In: *Engineering Applications of Artificial Intelligence*, Vol. 17, No. 4 (2004), pp. 337–344. DOI: `10.1016/j.engappai.2004.04.007` (cit. on p. 42).

[31] Rainer E Burkard, Stefan E Karisch, and Franz Rendl. “QAPLIB–a quadratic assignment problem library”. In: *Journal of Global optimization*, Vol. 10, No. 4 (1997), pp. 391–403. DOI: `10.1016/0377-2217(91)90197-4` (cit. on p. 44).

[32] EK Burke and AJ Smith. “Hybrid evolutionary techniques for the maintenance scheduling problem”. In: *IEEE transactions on power systems*, Vol. 15, No. 1 (2000), pp. 122–128. DOI: `10.1109/59.852110` (cit. on pp. 4, 44).

[33] Sébastien Cahon, Nordine Melab, and E-G Talbi. “Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics”. In: *Journal of heuristics*, Vol. 10, No. 3 (2004), pp. 357–380. DOI: `10.1023/B:HEUR.0000026900.92269.ec` (cit. on pp. 3, 37, 38, 47, 170).

[34] Erick Cantú-Paz. “A summary of research on parallel genetic algorithms”. In: *IlliGAL Report, University of Illinois.* (1997) (cit. on pp. 18, 19, 21).

[35] Erick Cantú-Paz et al. “A survey of parallel genetic algorithms”. In: *Calculateurs paralleles, reseaux et systems repartis*, Vol. 10, No. 2 (1998), pp. 141–171 (cit. on pp. 2, 3, 18–22, 49).

[36] Erick Cantú-Paz. *Efficient and accurate parallel genetic algorithms*. Vol. 1. (2000). DOI: `10.1007/978-1-4615-4369-5` (cit. on pp. 78, 82).

[37] Erick Cantú-Paz. “Topologies, migration rates, and multi-population parallel genetic algorithms”. In: *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1.* (1999), pp. 91–98 (cit. on p. 20).

[38] Erick Cantú-Paz and David E Goldberg. "Predicting speedups of idealized bounding cases of parallel genetic algorithms". In: *In Back, T.(Ed.), Proceedings of the Seventh International Conference on Genetic Algorithms*. (1996), pp. 113–121 (cit. on p. 20).

[39] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. "Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]". In: *IEEE Transactions on Neural Networks*, Vol. 20, No. 3 (2009), pp. 542–542. DOI: `10.1109/TNN.2009.2015974` (cit. on p. 30).

[40] Zhaomin Chen et al. "Autoencoder-based network anomaly detection". In: *2018 Wireless Telecommunications Symposium (WTS)*. (2018), pp. 1–5. DOI: `10.1109/WTS.2018.8363930` (cit. on p. 31).

[41] F. M. Cleveland. "IEC 61850-7-420 communications standard for distributed energy resources (DER)". In: *2008 IEEE Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century*. (2008), pp. 1–4. DOI: `10.1109/PES.2008.4596553` (cit. on pp. 108, 110).

[42] George F Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed systems: concepts and design*. (2005) (cit. on p. 27).

[43] Dipankar Dasgupta and Zbigniew Michalewicz. *Evolutionary algorithms in engineering applications*. (1997). DOI: `10.1007/978-3-662-03423-1` (cit. on p. 1).

[44] Luca Di Gaspero and Andrea Schaerf. "EasyLocal++: an object-oriented framework for the flexible design of local-search algorithms". In: *Software: Practice and Experience*, Vol. 33, No. 8 (2003), pp. 733–765. DOI: `10.1002/spe.524` (cit. on p. 47).

[45] Sergio Di Martino et al. "Towards migrating genetic algorithms for test data generation to the cloud". In: *Software Testing in the Cloud: Perspectives on an Emerging Discipline*. (2013), pp. 113–135. DOI: `10.4018/978-1-4666-2536-5.ch006` (cit. on pp. 3, 40, 47, 170).

[46] Andreas Drexl and Sigrid Knust. "Sports league scheduling: graph-and resource-based models". In: *Omega*, Vol. 35, No. 5 (2007), pp. 465–471. DOI: `10.1016/j.omega.2005.08.002` (cit. on p. 1).

[47] Joseph C Dunn. "A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters". In: *Journal of Cybernetics*. (1973). DOI: `10.1080/01969727308546046` (cit. on p. 143).

[48] Juan J. Durillo and Antonio J. Nebro. "jMetal: A Java framework for multi-objective optimization". In: *Advances in Engineering Software*, Vol. 42, No. 10 (2011), pp. 760–771. DOI: `https://doi.org/10.1016/j.advengsoft.2011.05.014` (cit. on p. 41).

[49] Michael Eder. "Hypervisor-vs. container-based virtualization". In: *Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM)*, Vol. 1 (2016). DOI: `10.1109/ICCV.2015.515` (cit. on p. 34).

[50] Thomas Erl. *Service-oriented architecture: concepts, technology, and design*. (2005) (cit. on p. 35).

[51] Patrick Th Eugster et al. "The many faces of publish/subscribe". In: *ACM computing surveys (CSUR)*, Vol. 35, No. 2 (2003), pp. 114–131. DOI: 10.1145/857076.857078 (cit. on p. 28).

[52] Pedro Fazenda, James McDermott, and Una May O'Reilly. "A Library to Run Evolutionary Algorithms in the Cloud using MapReduce". In: *European Conference on the Applications of Evolutionary Computation*. (2012), pp. 416–425. DOI: 10.1007/978-3-642-29178-4_42 (cit. on pp. 3, 41, 47, 170).

[53] Thomas A Feo and Mauricio GC Resende. "Greedy randomized adaptive search procedures". In: *Journal of global optimization*, Vol. 6, No. 2 (1995), pp. 109–133. DOI: 10.1007/BF01096763 (cit. on p. 44).

[54] Filomena Ferrucci, Pasquale Salza, and Federica Sarro. "Using Hadoop MapReduce for Parallel Genetic Algorithms: A Comparison of the Global, Grid and Island Models". In: *Evolutionary computation*, Vol. 26 (2017), pp. 535–567. DOI: 10.1162/evco_a_00213 (cit. on p. 40).

[55] Filomena Ferrucci et al. "A framework for genetic algorithms based on hadoop". In: *CoRR*. (2013). DOI: 10.48550/arXiv.1312.0086 (cit. on p. 40).

[56] Filomena Ferrucci et al. "A parallel genetic algorithms framework based on Hadoop MapReduce". In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. (2015), pp. 1664–1667. DOI: 10.1145/2695664.2696060 (cit. on p. 40).

[57] Andreas Fink and Stefan Voß. "HotFrame: a heuristic optimization framework". In: *Optimization software class libraries*. (2003), pp. 81–154. DOI: 10.1007/0-306-48126-X_4 (cit. on p. 47).

[58] David B Fogel. "An introduction to simulated evolutionary optimization". In: *IEEE transactions on neural networks*, Vol. 5, No. 1 (1994), pp. 3–14. DOI: 10.1109/72.265956 (cit. on p. 2).

[59] Gary B Fogel and David W Corne. *Evolutionary computation in bioinformatics*. (2003). DOI: 10.1016/B978-155860797-2/50000-7 (cit. on p. 1).

[60] Gianluigi Folino, Clara Pizzuti, and Giandomenico Spezzano. "Training distributed GP ensemble with a selective algorithm based on clustering and pruning for pattern classification". In: *IEEE Transactions on Evolutionary Computation*, Vol. 12, No. 4 (2008), pp. 458–468. DOI: 10.1109/TEVC.2007.906658 (cit. on p. 42).

[61] Gianluigi Folino and Giandomenico Spezzano. "P-cage: an environment for evolutionary computation in peer-to-peer systems". In: *European Conference on Genetic Programming*. (2006), pp. 341–350. DOI: 10.1007/11729976_31 (cit. on pp. 42, 170).

[62] Félix-Antoine Fortin et al. "DEAP: Evolutionary algorithms made easy". In: *The Journal of Machine Learning Research*, Vol. 13, No. 1 (2012), pp. 2171–2175. DOI: 10.5555/2503308.2503311 (cit. on pp. 3, 37, 38, 41, 43, 47, 170).

[63] Tobias Friedrich and Markus Wagner. “Seeding the initial population of multi-objective evolutionary algorithms: A computational study”. In: *Applied Soft Computing*, Vol. 33 (2015), pp. 223–230. DOI: 10.1016/j.asoc.2015.04.043 (cit. on p. 163).

[64] K Ganesh and M Punniyamoorthy. “Optimization of continuous–time production planning using hybrid genetic algorithms–simulated annealing”. In: *The International Journal of Advanced Manufacturing Technology*, Vol. 26, No. 1-2 (2005), pp. 148–154. DOI: 10.1007/s00170-003-1976-4 (cit. on p. 23).

[65] Mario Garcia-Valdez et al. “The evospace model for pool-based evolutionary algorithms”. In: *Journal of Grid Computing*, Vol. 13, No. 3 (2015), pp. 329–349. DOI: Garcia-Valdez (cit. on pp. 3, 41, 42).

[66] Mario García-Valdez and JJ Merelo. “evospace-js: asynchronous pool-based execution of heterogeneous metaheuristics”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion.* (2017), pp. 1202–1208. DOI: 10.1145/3067695.3082473 (cit. on pp. 3, 37, 42, 48, 170).

[67] David E Golberg. *Genetic algorithms in search, optimization, and machine learning.* 1989. DOI: 10.5555/534133 (cit. on p. 1).

[68] Yue-Jiao Gong et al. “Distributed evolutionary algorithms and their models: A survey of the state-of-the-art”. In: *Applied Soft Computing*, Vol. 34 (2015), pp. 286–300. DOI: 10.1016/j.asoc.2015.04.061 (cit. on pp. 20, 22, 47, 99).

[69] V Scott Gordon and Darrell Whitley. “Serial and parallel genetic algorithms as function optimizers”. In: *ICGA*. (1993), pp. 177–183. DOI: 10.5555/645513.657737 (cit. on p. 18).

[70] Martina Gorges-Schleuter. *Genetic algorithms and population structures: a massively parallel algorithm.* (1991) (cit. on p. 18).

[71] MOW Grond et al. “Multi-objective optimization techniques and applications in electric power systems”. In: *47th International Universities Power Engineering Conference (UPEC).* (2012), pp. 1–6. DOI: 10.1109/UPEC.2012.6398417 (cit. on p. 2).

[72] Crina Grosan and Ajith Abraham. “Hybrid evolutionary algorithms: methodologies, architectures, and reviews”. In: *Hybrid evolutionary algorithms.* (2007), pp. 1–17 (cit. on pp. 23, 43).

[73] Juan J Merelo Guervós and J Mario García-Valdez. “Introducing an event-based architecture for concurrent and distributed evolutionary algorithms”. In: *International Conference on Parallel Problem Solving from Nature.* (2018), pp. 399–410. DOI: 10.1007/978-3-319-99253-2_32 (cit. on pp. 3, 37, 41, 48, 170).

[74] Yuanxiong Guo, Yuguang Fang, and Pramod P Khargonekar. “Hierarchical Architecture for Distributed Energy Resource Management”. In: *Stochastic Optimization for Distributed Energy Resources in Smart Grids.* (2017), pp. 1–8. DOI: 10.1007/978-3-319-59529-0_1 (cit. on pp. 109, 110).

[75] Abdurrahman Hacioglu. “A novel usage of neural network in optimization and implementation to the internal flow systems”. In: *Aircraft Engineering and Aerospace Technology.* (2005). DOI: 10.1108/00022660510617095 (cit. on pp. 4, 23, 45, 46).

[76] Abdurrahman Hacioglu. “Fast evolutionary algorithm for airfoil design via neural network”. In: *AIAA journal*, Vol. 45, No. 9 (2007), pp. 2196–2203. DOI: 10.2514/1.24484 (cit. on pp. 4, 23, 45, 46).

[77] William E Hart, Natalio Krasnogor, and James E Smith. *Recent advances in memetic algorithms.* (2004). DOI: 10.1007/3-540-32363-5 (cit. on pp. 46, 110).

[78] Erez Hartuv and Ron Shamir. “A clustering algorithm based on graph connectivity”. In: *Information processing letters*, Vol. 76, No. 4-6 (2000), pp. 175–181. DOI: 10.1016/S0020-0190(00)00142-3 (cit. on pp. 30, 143).

[79] Christian Hirsch, Pradyumn Kumar Shukla, and Hartmut Schmeck. “Variable preference modeling using multi-objective evolutionary algorithms”. In: *International Conference on Evolutionary Multi-Criterion Optimization.* (2011), pp. 91–105. DOI: 10.1007/978-3-642-19893-9_7 (cit. on p. 13).

[80] Sepp Hochreiter and Jürgen Schmidhuber. “Long short–term memory”. In: *Neural computation*, Vol. 9, No. 8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735 (cit. on pp. 33, 149).

[81] Frank Hoffmeister and Thomas Bäck. “Genetic algorithms and evolution strategies: Similarities and differences”. In: *International conference on parallel problem solving from nature.* (1991), pp. 455–469. DOI: 10.1007/BFb0029787 (cit. on p. 11).

[82] John H Holland. “Outline for a Logical Theory of Adaptive Systems”. In: *Journal of the ACM (JACM)*, Vol. 9, No. 3 (1962), pp. 297–314. DOI: 10.1145/321127.321128 (cit. on pp. 13, 37).

[83] Zhexue Huang. “Extensions to the k-means algorithm for clustering large data sets with categorical values”. In: *Data mining and knowledge discovery*, Vol. 2, No. 3 (1998), pp. 283–304. DOI: 10.1023/A:1009769707641 (cit. on p. 30).

[84] J Hunger and Gottfried Huttner. “Optimization and analysis of force field parameters by combination of genetic algorithms and neural networks”. In: *Journal of Computational Chemistry*, Vol. 20, No. 4 (1999), pp. 455–471. DOI: 10.1002/(SICI)1096-987X(199903)20:4%3C455::AID-JCC6%3E3.0.CO;2-1 (cit. on pp. 4, 45).

[85] Jasmine Irani, Nitin Pise, and Madhura Phatak. “Clustering techniques and the similarity measures used in clustering: A survey”. In: *International journal of computer applications*, Vol. 134, No. 7 (2016), pp. 9–14. DOI: 10.5120/ijca2016907841 (cit. on p. 30).

[86] Wilfried Jakob. “A general cost-benefit-based adaptation framework for multimeme algorithms”. In: *Memetic Computing*, Vol. 2, No. 3 (2010), pp. 201–218. DOI: 10.1007/s12293-010-0040-9 (cit. on pp. 25, 46, 81).

[87] Wilfried Jakob. “Applying Evolutionary Algorithms Successfully: A Guide Gained from Real-world Applications”. In: *CoRR*. (2021). DOI: 10.48550/arXiv.2107.11300 (cit. on p. 14).

[88] Wilfried Jakob et al. “Fast multi-objective scheduling of jobs to constrained resources using a hybrid evolutionary algorithm”. In: *International Conference on Parallel Problem Solving from Nature*. (2008), pp. 1031–1040. DOI: 10.1007/978-3-540-87700-4_102 (cit. on pp. 16, 118).

[89] Wilfried Jakob et al. “Fast rescheduling of multiple workflows to constrained heterogeneous resources using multi-criteria memetic computing”. In: *Algorithms*, Vol. 6, No. 2 (2013), pp. 245–277. DOI: 10.3390/a6020245 (cit. on p. 118).

[90] Wilfried Jakob et al. “Towards coding strategies for forecasting-based scheduling in smart grids and the energy lab 2.0”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. (2017), pp. 1271–1278. DOI: 10.1145/3067695.3082481 (cit. on pp. 25, 26, 45, 118).

[91] Chao Jin, Christian Vecchiola, and Rajkumar Buyya. “MRPGA: an extension of MapReduce for parallelizing genetic algorithms”. In: *2008 IEEE Fourth International Conference on eScience*. (2008), pp. 214–221. DOI: 10.1109/eScience.2008.78 (cit. on pp. 3, 39, 47, 170).

[92] Yaochu Jin. “A comprehensive survey of fitness approximation in evolutionary computation”. In: *Soft computing*, Vol. 9, No. 1 (2005), pp. 3–12. DOI: 10.1007/s00500-003-0328-5 (cit. on p. 43).

[93] Krzysztof Jurczuk, Marcin Czajkowski, and Marek Kretowski. “Evolutionary induction of a decision tree for large-scale data: a GPU-based approach”. In: *Soft Computing*, Vol. 21, No. 24 (2017), pp. 7363–7379. DOI: 10.1007/s00500-016-2280-1 (cit. on pp. 43, 47, 170).

[94] Krzysztof Jurczuk, Marcin Czajkowski, and Marek Kretowski. “Multi-GPU approach for big data mining: global induction of decision trees”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. (2019), pp. 175–176. DOI: 10.1007/978-3-319-99259-4_37 (cit. on pp. 43, 47, 170).

[95] Krzysztof Jurczuk, Daniel Reska, and Marek Kretowski. “What are the limits of evolutionary induction of decision trees?” In: *International Conference on Parallel Problem Solving from Nature*. (2018), pp. 461–473 (cit. on pp. 43, 47, 170).

[96] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. “Reinforcement learning: A survey”. In: *Journal of artificial intelligence research*, Vol. 4 (1996), pp. 237–285. DOI: 10.1613/jair.301 (cit. on p. 30).

[97] Edward Keedwell and Soon-Thiam Khu. “A hybrid genetic algorithm for the design of water distribution networks”. In: *Engineering Applications of Artificial Intelligence*, Vol. 18, No. 4 (2005), pp. 461–472. DOI: 10.1016/j.engappai.2004.10.001 (cit. on pp. 4, 163).

[98] Maarten Keijzer et al. “Evolving objects: A general purpose evolutionary computation library”. In: *International Conference on Artificial Evolution (Evolution Artificielle).* (2001), pp. 231–242. DOI: 10.1007/3-540-46033-0_19 (cit. on p. 38).

[99] Hatem Khalloof et al. “A distributed modular scalable and generic framework for parallelizing population-based metaheuristics”. In: *International Conference on Parallel Processing and Applied Mathematics*. (2019), pp. 432–444. DOI: 10.1007/978-3-030-43229-4_37 (cit. on p. 50).

[100] Hatem Khalloof et al. “A generic distributed microservices and container based framework for metaheuristic optimization”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. (2018), pp. 1363–1370. DOI: 10.1145/3205651.3208253 (cit. on p. 50).

[101] Hatem Khalloof et al. “A Generic Flexible and Scalable Framework for Hierarchical Parallelization of Population-Based Metaheuristics”. In: *Proceedings of the 12th International Conference on Management of Digital EcoSystems*. (2020), pp. 124–131. DOI: 10.1145/3415958.3433041 (cit. on p. 100).

[102] Hatem Khalloof et al. “A generic flexible and scalable framework for hierarchical parallelization of population-based metaheuristics”. In: *Internet of Things*, Vol. 16 (2021), p. 100433. DOI: 10.1016/j.iot.2021.100433 (cit. on p. 100).

[103] Hatem Khalloof et al. “A Generic Scalable Method for Scheduling Distributed Energy Resources Using Parallelized Population-Based Metaheuristics”. In: *Proceedings of the Future Technologies Conference*. (2020), pp. 1–21. DOI: 10.1007/978-3-030-63089-8_1 (cit. on p. 100).

[104] Hatem Khalloof et al. “Facilitating the hybridization of parallel evolutionary algorithms in cluster computing environments”. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. (2022), pp. 2001–2008. DOI: 10.1145/3520304.3533997 (cit. on p. 136).

[105] Hatem Khalloof et al. “Superlinear Speedup of Parallel Population-Based Metaheuristics: A Microservices and Container Virtualization Approach”. In: *International Conference on Intelligent Data Engineering and Automated Learning*. (2019), pp. 386–393. DOI: 10.1007/978-3-030-33607-3_42 (cit. on p. 50).

[106] Memoona Khanum et al. “A survey on unsupervised machine learning algorithms for automation, classification and maintenance”. In: *International Journal of Computer Applications*, Vol. 119, No. 13 (2015). DOI: 10.5120/21131-4058 (cit. on pp. 29, 139).

[107] Udo Kohlmorgen, Hartmut Schmeck, and Knut Haase. “Experiences with fine-grained parallel genetic algorithms”. In: *Annals of Operations Research*, Vol. 90 (1999), pp. 203–219. DOI: 10.1023/A:1018912715283 (cit. on p. 20).

[108] Oliver Kramer. “Genetic algorithms”. In: *Genetic algorithm essentials*. (2017), pp. 11–19. DOI: 10.1007/978-3-319-52156-5_2 (cit. on p. 15).

[109] Anders Krogh. "What are artificial neural networks?" In: *Nature biotechnology*, Vol. 26, No. 2 (2008), pp. 195–197. DOI: 10.1038/nbt1386 (cit. on pp. 30, 31).

[110] Yiu-Wing Leung and Yuping Wang. "An orthogonal genetic algorithm with quantization for global numerical optimization". In: *IEEE Transactions on Evolutionary computation*, Vol. 5, No. 1 (2001), pp. 41–53. DOI: 10.1109/4235.910464 (cit. on pp. 4, 45).

[111] Rhydian Lewis. "A survey of metaheuristic-based techniques for university timetabling problems". In: *OR spectrum*, Vol. 30, No. 1 (2008), pp. 167–190. DOI: 10.1007/s00291-007-0097-0 (cit. on p. 1).

[112] F Li, R Morgan, and D Williams. "Hybrid genetic approaches to ramping rate constrained dynamic economic dispatch". In: *Electric power systems research*, Vol. 43, No. 2 (1997), pp. 97–103. DOI: 10.1016/S0378-7796(97)01165-6 (cit. on p. 4).

[113] Hepeng Li et al. "A genetic algorithm-based hybrid optimization approach for microgrid energy management". In: *2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*. (2015), pp. 1474–1478. DOI: 10.1109/CYBER.2015.7288162 (cit. on p. 110).

[114] HZ Liang and HB Gooi. "Unit commitment in microgrids by improved genetic algorithm". In: *2010 Conference Proceedings IPEC*. (2010), pp. 842–847. DOI: 10.1109/IPECON.2010.5697083 (cit. on p. 110).

[115] Dudy Lim et al. "Efficient hierarchical parallel genetic algorithms using grid computing". In: *Future Generation Computer Systems*, Vol. 23, No. 4 (2007), pp. 658–670. DOI: 10.1016/j.future.2006.10.008 (cit. on pp. 42, 170).

[116] Shyh-Chang Lin, Erik D Goodman, and William F Punch. "Investigating parallel genetic algorithms on job shop scheduling problems". In: *International Conference on Evolutionary Programming*. (1997), pp. 383–393. DOI: 10.1007/BFb0014827 (cit. on p. 42).

[117] Shyh-Chang Lin, William F. Punch, and Erik D. Goodman. "Coarse-grain parallel genetic algorithms: categorization and new approach". In: *Proceedings of 1994 6th IEEE Symposium on Parallel and Distributed Processing*. (1994), pp. 28–37. DOI: 10.1109/SPDP.1994.346184 (cit. on pp. 18, 19).

[118] Yung-Chien Lin, Kao-Shing Hwang, and Feng-Sheng Wang. "A mixed-coding scheme of evolutionary algorithms to solve mixed-integer nonlinear programming problems". In: *Computers&Mathematics with Applications*, Vol. 47, No. 8-9 (2004), pp. 1295–1307. DOI: 10.1016/S0898-1221(04)90123-X (cit. on p. 23).

[119] Xavier Llora et al. "Meandre: Semantic-driven data-intensive flows in the clouds". In: *2008 IEEE Fourth International Conference on eScience*. (2008), pp. 238–245. DOI: 10.1109/eScience.2008.172 (cit. on p. 41).

[120] Sushil J Louis. "Working from blueprints: evolutionary learning for design". In: *Artificial Intelligence in Engineering*, Vol. 11, No. 3 (1997), pp. 335–341. DOI: 10.1016/S0954-1810(96)00048-9 (cit. on pp. 4, 44, 163).

[121] Thé Van Luong, Nouredine Melab, and El-Ghazali Talbi. “GPU-based island model for evolutionary algorithms”. In: *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. (2010), pp. 1089–1096. DOI: 10.1145/1830483.1830685 (cit. on pp. 43, 47, 170).

[122] Gabriel Luque and Enrique Alba. *Parallel genetic algorithms: theory and real world applications*. Vol. 367. (2011). DOI: 10.1007/978-3-642-22084-5 (cit. on pp. 16, 18–21, 78).

[123] Heikki Maaranen, Kaisa Miettinen, and Antti Penttinen. “On initial populations of a genetic algorithm for continuous optimization problems”. In: *Journal of Global Optimization*, Vol. 37, No. 3 (2007), p. 405. DOI: 10.1007/s10898-006-9056-6 (cit. on p. 43).

[124] Laurent Magnier and Fariborz Haghighat. “Multiobjective optimization of building design using TRNSYS simulations, genetic algorithm, and Artificial Neural Network”. In: *Building and Environment*, Vol. 45, No. 3 (2010), pp. 739–746. DOI: 10.1016/j.buildenv.2009.08.016 (cit. on pp. 4, 23, 45, 46).

[125] Alfonso C Martínez-Estudillo et al. “Hybridization of evolutionary algorithms and local search by means of a clustering method”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, Vol. 36, No. 3 (2006), pp. 534–545. DOI: 10.1109/tsmcb.2005.860138 (cit. on p. 46).

[126] Suraya Masrom et al. “Software Framework for Flexible User Defined Metaheuristic Hybridization”. In: *International Conference on Advanced Software Engineering and Its Applications*. (2010), pp. 218–227. DOI: 10.1007/978-3-642-17578-7_22 (cit. on p. 47).

[127] Ingo Mauser et al. “Adaptive building energy management with multiple commodities and flexible evolutionary optimization”. In: *Renewable Energy*, Vol. 87 (2016), pp. 911–921. DOI: 10.1016/j.renene.2015.09.003 (cit. on p. 110).

[128] Ingo Mauser et al. “Encodings for evolutionary algorithms in smart buildings with energy management systems”. In: *2014 IEEE Congress on Evolutionary Computation (CEC)*. (2014), pp. 2361–2366. DOI: 10.1109/CEC.2014.6900633 (cit. on p. 110).

[129] Larry R Medsker and LC Jain. “Recurrent neural networks”. In: *Design and Applications*, Vol. 5 (1999), pp. 64–67. DOI: 10.1201/9781420049176 (cit. on p. 32).

[130] Juan J Merelo et al. “Sofea: A pool-based framework for evolutionary algorithms using couchdb”. In: *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*. (2012), pp. 109–116. DOI: 10.1145/2330784.2330802 (cit. on pp. 3, 43, 48, 170).

[131] K Meri et al. “Cloud-based evolutionary algorithms: An algorithmic study”. In: *Natural Computing*, Vol. 12, No. 2 (2013), pp. 135–147. DOI: 10.1007/s11047-012-9358-1 (cit. on pp. 3, 43, 48, 170).

[132] Zbigniew Michalewicz. *Genetic algorithms + data structures= evolution programs*. (1996). DOI: 10.1007/978-3-662-03315-9 (cit. on p. 4).

[133] Martin Middendorf et al. "An evolutionary approach to dynamic task scheduling on FPGAs with restricted buffer". In: *Journal of Parallel and Distributed Computing*, Vol. 62, No. 9 (2002), pp. 1407–1420. DOI: 10.1006/jpdc.2002.1853 (cit. on p. 13).

[134] Glenn W Milligan and Martha C Cooper. "Methodology review: Clustering methods". In: *Applied psychological measurement*, Vol. 11, No. 4 (1987), pp. 329–354. DOI: 10.1177/014662168701100401 (cit. on p. 30).

[135] Tom M Mitchell et al. *Machine learning*. (1997) (cit. on p. 29).

[136] Antonio J Nebro et al. "Design and architecture of the jMetalSP framework". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. (2017), pp. 1239–1246. DOI: 10.1145/3067695.3082466 (cit. on pp. 3, 41, 47, 170).

[137] Mohsen Nemati, Martin Braun, and Stefan Tenbohlen. "Optimization of unit commitment and economic dispatch in microgrids based on genetic algorithm and mixed integer linear programming". In: *Applied energy*, Vol. 210 (2018), pp. 944–963. DOI: 10.1016/j.apenergy.2017.07.007 (cit. on p. 110).

[138] Mohsen Nemati et al. "Optimization of microgrids short term operation based on an enhanced genetic algorithm". In: *2015 IEEE Eindhoven PowerTech*. (2015), pp. 1–6. DOI: 10.1109/PTC.2015.7232801 (cit. on p. 110).

[139] Sam Newman. *Building microservices: designing fine-grained systems*. (2015) (cit. on pp. 28, 29, 50).

[140] Susan Offner. "Mendel's peas & the nature of the gene: genes code for proteins & proteins determine phenotype". In: *The american biology Teacher*, Vol. 73, No. 7 (2011), pp. 382–387. DOI: 10.1525/abt.2011.73.7.3 (cit. on p. 14).

[141] FY Osisanwo et al. "Supervised machine learning algorithms: classification and comparison". In: *International Journal of Computer Trends and Technology (IJCTT)*, Vol. 48, No. 3 (2017), pp. 128–138. DOI: 10.14445/22312803/IJCTT-V48P126 (cit. on p. 29).

[142] Ibrahim H Osman and Gilbert Laporte. *Metaheuristics: A bibliography*. (1996). DOI: 10.1007/BF02125421 (cit. on p. 1).

[143] Ibrahim Hassan Osman. "Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem". In: *Annals of operations research*, Vol. 41, No. 4 (1993), pp. 421–451. DOI: 10.1007/BF02023004 (cit. on pp. 1, 50).

[144] K Rajmohan Padiyar and Anil M Kulkarni. *Dynamics and control of electric transmission and microgrids*. (2019). DOI: 10.1002/9781119173410 (cit. on pp. 108, 110).

[145] Ben Paechter et al. "A distributed resource evolutionary algorithm machine (DREAM)". In: *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*. Vol. 2. (2000), pp. 951–958. DOI: 10.1109/CEC.2000.870746 (cit. on pp. 3, 37, 38, 47, 170).

[146] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. (1998). DOI: 10.1007/BF02023004 (cit. on pp. 1, 13).

[147] Walter Hugo Lopez Pinaya et al. "Autoencoders". In: *Machine learning*. (2020), pp. 193–208. DOI: 10.1016/B978-0-12-815739-8.00011-0"s (cit. on p. 31).

[148] Shahryar Rahnamayan, Hamid R Tizhoosh, and Magdy MA Salama. "Opposition–based differential evolution". In: *IEEE Transactions on Evolutionary computation*, Vol. 12, No. 1 (2008), pp. 64–79. DOI: 10.1109/TEVC.2007.894200 (cit. on p. 45).

[149] De Rainville et al. "DEAP: enabling nimbler evolutions". In: *ACM SIGEVOlution*, Vol. 6, No. 2 (2014), pp. 17–26. DOI: 10.1145/2597453.2597455 (cit. on p. 39).

[150] Abdur Rais and Ana Viana. "Operations research in healthcare: a survey". In: *International transactions in operational research*, Vol. 18, No. 1 (2011), pp. 1–31. DOI: 10.1111/j.1475-3995.2010.00767.x (cit. on p. 1).

[151] Aurora Ramirez et al. "JCLEC-MO: a Java suite for solving many-objective optimization engineering problems". In: *Engineering Applications of Artificial Intelligence*, Vol. 81 (2019), pp. 14–28. DOI: 10.1016/j.engappai.2019.02.003 (cit. on p. 39).

[152] L A Rastrigin. "Systems of extremal control". In: *Nauka.* (1974) (cit. on p. 85).

[153] Elizabeth L Ratnam et al. "Residential load and rooftop PV generation: an Australian distribution network dataset". In: *International Journal of Sustainable Energy*, Vol. 36, No. 8 (2017), pp. 787–806. DOI: 10.1080/14786451.2015.1100196 (cit. on pp. 108, 111, 113, 114, 141).

[154] Ingo Rechenberg. "Evolutionsstrategien". In: *Simulationsmethoden in der Medizin und Biologie*. (1978), pp. 83–114. DOI: 10.1007/978-3-642-81283-5_8 (cit. on pp. 13, 37).

[155] Rommel G Regis and Christine A Shoemaker. "Local function approximation in evolutionary algorithms for the optimization of costly functions". In: *IEEE transactions on evolutionary computation*, Vol. 8, No. 5 (2004), pp. 490–505. DOI: 10.1109/TEVC.2004.835247 (cit. on pp. 4, 45).

[156] Rebecca Rivers, Alex R Bertels, and Daniel R Tauritz. "Asynchronous parallel evolutionary algorithms: Leveraging heterogeneous fitness evaluation times for scalability and elitist parsimony pressure". In: *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*. (2015), pp. 1429–1430. DOI: 10.1145/2739482.2764718 (cit. on p. 19).

[157] A Rodríguez del Nozal et al. “Application of genetic algorithms for unit commitment and economic dispatch problems in microgrids”. In: *Nature Inspired Computing for Data Science.* (2020), pp. 139–167. DOI: 10.1007/978-3-030-33820-6_6 (cit. on pp. 108, 111).

[158] Franz Rothlauf. “Optimization Methods”. In: *Design of Modern Heuristics: Principles and Application.* (2011), pp. 45–102. DOI: 10.1007/978-3-540-72962-4_3 (cit. on p. 13).

[159] Franz Rothlauf. “Representations for genetic and evolutionary algorithms”. In: *Representations for Genetic and Evolutionary Algorithms.* (2006), pp. 9–32. DOI: 10.1007/3-540-32444-5_2 (cit. on p. 14).

[160] Peter J Rousseeuw. “Silhouettes: a graphical aid to the interpretation and validation of cluster analysis”. In: *Journal of computational and applied mathematics*, Vol. 20 (1987), pp. 53–65. DOI: 10.1016/0377-0427(87)90125-7 (cit. on p. 143).

[161] Gautam Roy et al. “A distributed pool architecture for genetic algorithms”. In: *2009 IEEE Congress on Evolutionary Computation.* (2009), pp. 1177–1184. DOI: 10.1007/s11047-012-9358-1 (cit. on pp. 3, 43, 48, 170).

[162] Carolina Salto et al. “Developing Genetic Algorithms Using Different MapReduce Frameworks: MPI vs. Hadoop”. In: *Conference of the Spanish Association for Artificial Intelligence.* (2018), pp. 262–272. DOI: 10.1007/978-3-030-00374-6_25 (cit. on p. 47).

[163] Pasquale Salza and Filomena Ferrucci. “An approach for parallel genetic algorithms in the cloud using software containers”. In: *CoRR.*, abs/1606.06961 (2016) (cit. on pp. 3, 37, 41, 48, 170).

[164] Pasquale Salza and Filomena Ferrucci. “Speed up genetic algorithms in the cloud using software containers”. In: *Future Generation Computer Systems*, Vol. 92 (2019), pp. 276–289. DOI: 10.1016/j.future.2018.09.066 (cit. on pp. 3, 37, 41, 48, 170).

[165] Pasquale Salza, Filomena Ferrucci, and Federica Sarro. “elephant56: Design and implementation of a parallel genetic algorithms framework on hadoop MapReduce”. In: *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion.* (2016), pp. 1315–1322. DOI: 10.1145/2908961.2931722 (cit. on pp. 3, 40, 47, 170).

[166] Jeffrey R Sampson. *Adaptation in natural and artificial systems (John H. Holland).* (1976). DOI: 10.1137/1018105 (cit. on p. 37).

[167] Frank Schlottmann and Detlef Seese. “A hybrid heuristic approach to discrete multi-objective optimization of credit portfolios”. In: *Computational statistics&data analysis*, Vol. 47, No. 2 (2004), pp. 373–399. DOI: 10.1016/j.csda.2003.11.016 (cit. on pp. 23, 46).

[168] Rüdiger Schollmeier. “A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications”. In: *Proceedings First International Conference on Peer-to-Peer Computing*. (2001), pp. 101–102. DOI: 10.1109/P2P.2001.990434 (cit. on p. 42).

[169] Hans-Paul Schwefel. *Numerical optimization of computer models*. (1982). DOI: 10.1057/jors.1982.238 (cit. on p. 37).

[170] Dylan Sherry et al. “Flex-GP: genetic programming on the cloud”. In: *European Conference on the Applications of Evolutionary Computation*. (2012), pp. 477–486. DOI: 10.1007/978-3-642-29178-4_48 (cit. on pp. 3, 41, 47, 170).

[171] SN Sivanandam and SN Deepa. “Genetic algorithms”. In: *Introduction to genetic algorithms*. (2008), pp. 15–37. DOI: 10.1007/978-3-540-73190-0_2 (cit. on p. 18).

[172] Stephen Soltesz et al. “Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors”. In: *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*. (2007), pp. 275–287. DOI: 10.1145/1272996.1273025 (cit. on p. 34).

[173] Anjan Kumar Swain and Alan S Morris. “A novel hybrid evolutionary programming method for function optimization”. In: *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No. 00TH8512)*. Vol. 1. (2000), pp. 699–705. DOI: 10.1109/CEC.2000.870366 (cit. on pp. 4, 23, 46).

[174] El-Ghazali Talbi. *Metaheuristics: from design to implementation*. Vol. 74. (2009). DOI: 10.5555/1718024 (cit. on p. 2).

[175] Hamid R Tizhoosh. “Opposition-based learning: a new scheme for machine intelligence”. In: *International conference on computational intelligence for modelling, control and automation and international conference on intelligent agents, web technologies and internet commerce (CIMCA-IAWTIC’06)*. Vol. 1. (2005), pp. 695–701. DOI: 10.1109/4235.910464 (cit. on pp. 4, 45, 163).

[176] CL Tseng. “On power system generation unit commitment problems. Berkeley: University of California”. PhD thesis. Doctoral Thesis, (1996) (cit. on p. 110).

[177] Lin-Yu Tseng and Shyi-Ching Liang. “A hybrid metaheuristic for the quadratic assignment problem”. In: *Computational Optimization and Applications*, Vol. 34, No. 1 (2006), pp. 85–113. DOI: 10.1007/s10589-005-3069-9 (cit. on pp. 4, 43).

[178] James Turnbull. *The Docker Book: Containerization is the new virtualization*. (2014) (cit. on p. 34).

[179] Maarten Van Steen and A Tanenbaum. “Distributed systems principles and paradigms”. In: *Network*, Vol. 2 (2002), p. 28 (cit. on p. 27).

[180] Manuel Vázquez and L Darrell Whitley. “A hybrid genetic algorithm for the quadratic assignment problem”. In: *Proceedings of the 2nd annual conference on genetic and evolutionary computation*. (2000), pp. 135–142. DOI: 10.5555/2933718.2933737 (cit. on pp. 4, 44).

[181] Sebastián Ventura et al. “JCLEC: a Java framework for evolutionary computation”. In: *Soft computing*, Vol. 12, No. 4 (2008), pp. 381–392. DOI: 10.1007/s00500-007-0172-0 (cit. on pp. 37, 39, 47, 170).

[182] Subhashini Venugopalan et al. “Sequence to sequence-video to text”. In: *Proceedings of the IEEE international conference on computer vision*. (2015), pp. 4534–4542 (cit. on p. 33).

[183] Abhishek Verma, David E Goldberg, and Roy H Campbell. “Scaling Simple and Compact Genetic Algorithms using MapReduce”. In: *Ninth International Conference on Intelligent Systems Design and Applications (ISDA)*. (2009), pp. 13–18. DOI: 10.1.1.173.2387 (cit. on pp. 3, 40, 47, 170).

[184] Christos Voudouris et al. “iOpt: A software toolkit for heuristic search methods”. In: *International Conference on Principles and Practice of Constraint Programming*. (2001), pp. 716–729. DOI: 10.1007/3-540-45578-7_58 (cit. on p. 47).

[185] Stefan Wagner and Michael Affenzeller. “Heuristiclab: A generic and extensible optimization environment”. In: *Adaptive and Natural Computing Algorithms*. (2005), pp. 538–541. DOI: 10.1007/3-211-27389-1_130 (cit. on p. 47).

[186] Ling Wang. “A hybrid genetic algorithm–neural network strategy for simulation optimization”. In: *Applied Mathematics and Computation*, Vol. 170, No. 2 (2005), pp. 1329–1343. DOI: 10.1016/j.amc.2005.01.024 (cit. on pp. 4, 23, 45).

[187] Darrell Whitley. “A genetic algorithm tutorial”. In: *Statistics and computing*, Vol. 4, No. 2 (1994), pp. 65–85. DOI: 10.1007/BF00175354 (cit. on p. 13).

[188] David H Wolpert and William G Macready. “No free lunch theorems for optimization”. In: *IEEE transactions on evolutionary computation*, Vol. 1, No. 1 (1997), pp. 67–82. DOI: 10.1109/4235.585893 (cit. on pp. 4, 135).

[189] Matei Zaharia et al. “Spark: Cluster computing with working sets”. In: *HotCloud*, Vol. 10, No. 10 (2010), p. 95. DOI: 10.5555/1863103.1863113 (cit. on p. 41).

[190] Jun Zhang et al. “Evolutionary computation meets machine learning: A survey”. In: *IEEE Computational Intelligence Magazine*, Vol. 6, No. 4 (2011), pp. 68–75. DOI: 10.1007/978-3-540-73297-6_1 (cit. on pp. 23, 43).

[191] Qingfu Zhang and Yiu-Wing Leung. “An orthogonal genetic algorithm for multimedia multicast routing”. In: *IEEE Transactions on Evolutionary Computation*, Vol. 3, No. 1 (1999), pp. 53–62. DOI: 10.1109/4235.752920 (cit. on p. 44).

[192] Zongzhao Zhou et al. “A study on polynomial regression and Gaussian process global surrogate model in hierarchical surrogate-assisted evolutionary algorithm”. In: *2005 IEEE congress on evolutionary computation*. Vol. 3. (2005), pp. 2832–2839. DOI: 10.1109/CEC.2005.1555050 (cit. on pp. 4, 45).

# A. List of Publications

**Journal Articles**

1. Hatem Khalloof et al. “A generic flexible and scalable framework for hierarchical parallelization of population-based metaheuristics”. In: Internet of Things, Vol. 16 (2021), p. 100433. DOI: 10.1016/j.iot.2021.100433

**Conference Articles**

1. Hatem Khalloof et al. “A generic distributed microservices and container based framework for metaheuristic optimization”. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. (2018), pp. 1363–1370. DOI: 10.1145/3205651.3208253
2. Hatem Khalloof et al. “A distributed modular scalable and generic framework for parallelizing population-based metaheuristics”. In: International Conference on Parallel Processing and Applied Mathematics. (2019), pp. 432–444. DOI: 10.1007/978-3-030-43229-4_37
3. Hatem Khalloof et al. “Superlinear Speedup of Parallel Population-Based Metaheuristics: A Microservices and Container Virtualization Approach”. In: International Conference on Intelligent Data Engineering and Automated Learning. (2019), pp. 386–393. DOI: 10.1007/978-3-030-33607-3_42
4. Hatem Khalloof et al. “A Generic Flexible and Scalable Framework for Hierarchical Parallelization of Population-Based Metaheuristics”. In: Proceedings of the 12th International Conference on Management of Digital EcoSystems. (2020), pp. 124–131. DOI: 10.1145/3415958.3433041
5. Hatem Khalloof et al. “A Generic Scalable Method for Scheduling Distributed Energy Resources Using Parallelized Population-Based Metaheuristics”. In: Proceedings of the Future Technologies Conference. (2020), pp. 1–21. DOI: 10.1007/ 978-3-030-63089-8_1
6. Hatem Khalloof et al. “Facilitating the hybridization of parallel evolutionary algorithms in cluster computing environments”. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. (2022), pp. 2001–2008. DOI: 10.1145/3520304.3533997