Application of Evolutionary Optimization in Task Mapping and Scheduling for Heterogeneous Mobile-Edge Computing

Nikolina Frid, Marko Đurasević
University of Zagreb Faculty of Electrical Engineering and Computing, Zagreb, Croatia
nikolina.frid@fer.hr

***Abstract* — The growing demand for computational power in mobile and Internet of Things (IoT) applications, particularly in real-time contexts like autonomous driving, requires efficient task offloading to the cloud. However, challenges arise due to the need for stable internet connectivity and uncertainties in cloud responsiveness, especially in real-time systems. To overcome these challenges, partial task offloading to edge devices, which are placed between mobile or IoT devices and the cloud, is employed. This paper investigates task mapping and scheduling within a heterogeneous mobile-edge-cloud architecture with limited connectivity between nodes, and constrained task executability at the mobile and edge layers. Based on similarities with challenges in heterogeneous multiprocessor embedded systems, the paper explores the application of NSGA-II-based algorithms that were previously successfully applied in task mapping and scheduling for sparsely connected heterogeneous multiprocessor platforms. The algorithms are evaluated in the heterogeneous mobile-edge-cloud setting, and based on their results, possibilities for optimizing computational task allocation are discussed.**

Key words – mobile-edge computing, evolutionary algorithms, optimization, scheduling

I. INTRODUCTION

The classical cloud architecture is increasingly being replaced by the Mobile Edge Cloud (MEC) architecture, primarily due to numerous IoT systems that require data processing closer to the data source [1]-[3]. MEC applications span across various sectors, such as healthcare, augmented and virtual reality, multi-player gaming, interactive multimedia, video analytics, smart environments, industrial control systems, vehicular communications, and road traffic monitoring [4]. However, a significant problem arises as data generated by mobile devices, wearables, sensors, and IoT devices is often sent to remote clouds for processing and storage [5][6]. With the anticipated data generation from IoT devices projected to reach 79.4 zettabytes by 2025 [7], traditional cloud architecture can impair applications and degrade Quality of Service (QoS).

The challenge of processing time (real-time or near-real-time, as in autonomous driving [8]) and data transfer remains crucial due to its impact on costs and energy consumption, particularly for battery-powered IoT devices. The solution involves bringing computing resources closer to devices and sensors, potentially distributing them across Mobile/IoT, edge, and cloud layers [9]. However, edge resources differ from cloud ones, being resource-constrained, heterogeneous, and subject to varying workloads.

Managing resources in fog and edge computing poses a significant challenge, as traditional heuristic approaches struggle with diverse and dynamic workloads [10]-[12]. Hence, the utilization of AI/ML techniques to optimize resource management becomes necessary [13]-[15]. In this context, the problem of task mapping and scheduling can be compared to the mapping and scheduling problem in heterogeneous embedded computing systems [16][17]. Therefore, this paper aims to examine several algorithms that have proven successful in that domain and explore the possibilities and limitations of their application in mobile-edge-cloud architecture.

The rest of this paper is organized as follows. Section 2 reviews the relevant literature on scheduling and mapping in both MEC architecture and heterogeneous multiprocessor embedded systems. In Section 3, we define the problem, covering task and platform models, along with the optimization problem formulation and algorithm descriptions. Section 4 provides the evaluation, including test cases, results, and discussions. The paper concludes in Section 5, summarizing key insights and contributions.

II. RELATED WORK

Task mapping and scheduling within the Mobile Edge Cloud architecture is an active research area. Typically, researchers optimize the total task execution time, deadline violation, and energy consumption [18]. However, due to the complexity of this problem, many studies tend to simplify the inherent heterogeneity of this architecture by modeling the edge and the cloud as a single entity, overlooking its individual nodes [19].

Moreover, existing works often assume a simplified connection model where all elements are fully interconnected and between each two elements there is a single connection, without exploring the possibilities of multiple connections between two elements with varying speeds or the lack of connection between two elements [20]-[22]. Additionally, these studies tend to make assumptions such as considering the mobile component incapable of executing any tasks [20][21], or assuming that every node can execute all tasks, albeit at different speeds [19][22]. However, none of these works consider the case where an

element is able to execute some but not all tasks from the taskset.

The study conducted in [23] tackles the issue of task mapping and scheduling for clustered heterogeneous multiprocessor embedded platforms characterized by limitations in execution capabilities and connectivity. These limitations manifest in scenarios where certain processors are unable to execute specific tasks, and not every pair of processors is interconnected. Consequently, these constraints give rise to infeasible solutions within the design space, wherein a solution is deemed infeasible when a task is mapped to a processor either incapable of executing the task or disconnected from processors where the task's predecessors or successors are executed. In this study, several new algorithms based on the Non-dominated Sorting Genetic Algorithm II (NSGA-II) meta-heuristic [24] are introduced and achieve nearly 100% success rate in finding feasible solutions. Due to the significant constraints in such a problem caused by reduced connectivity and the incapability of heterogeneous elements to execute all tasks, which results in the very large number of infeasible solutions in the design space, finding a feasible solution is a success in itself [25][26].

As stated earlier, parallels can be drawn between mapping and scheduling problems in the embedded heterogeneous systems and the Mobile Edge Cloud architecture. Hence, this paper will explore how these algorithms can be applied to a Mobile Edge Cloud architecture in which there are different nodes, also grouped in clusters, and there is a certain limit to interconnectivity, albeit a little more relaxed than in the case explored for embedded systems described in the original paper.

III. METHODOLOGY

In this section a comprehensive problem definition that encompasses application models, platform models, and the formulation of the optimization problem is presented. Furthermore, within this section, the algorithms applied to the scheduling and mapping problem within the context of Mobile Edge Cloud architecture are described.

A. Problem definition

The application which is executed on the MEC architecture is modelled as a set of tasks connected in a Directed Acyclic Graph (DAG), $AG = (T, Ch)$. Graph vertices $t \in T$ represent application tasks, while edges connected to each vertex t, $Ch_t \subseteq Ch$ represent communication between tasks. Edge weight, given by a weight function:

$$w_{Ch}: Ch \to \mathbb{R} \quad (1)$$

is the amount of data exchanged (in KB) between two tasks.

The platform architecture is modelled as an undirected graph $PG = (P, L)$. Node $p \in P$ represents a processing element and $l \in L$ represents a link between two processors. The weight on an edge, given by a weight function:

$$w_l : L \to \mathbb{R} \quad (2)$$

represents communication speed between two processors in both directions (read and write operations are assumed to be the same). For every p, a set of execution times of each task from AG is defined as $T_p = \{T_0, \ldots T_n\}$. If a task cannot be executed on a certain processing element, then its execution time is set to infinity. It is assumed that a processor can execute one task at a time, and there is no pre-emption.

The platform is generally divided into three layers: Mobile/IoT, Edge, and Cloud, as shown in Figure 1. Accordingly, we consider three main types of nodes. Mobile/IoT node represents a single IoT or mobile phone node that represents the endpoint from which the taskset originates and that collects the results after the entire task set has finished execution. It is assumed that this node type can have certain limitations in its ability to execute certain tasks from the taskset. For example, certain tasks might be implemented using specific libraries (e.g. signal operations, or graphic manipulations) that cannot be executed on general purpose mobile and embedded processors without significant redesign [26][27].

In the Edge layer, there is a certain number of edge nodes. Each edge node can internally consist of multiple cores. Cores inside the edge node are fully interconnected with a high-speed connection. Edge nodes are connected to Mobile/IoT nodes, to each other, and to the Cloud. Each of these connections can have different throughout. In general, connections to Cloud and adjacent edge nodes are considered to be faster than connections to Mobile/IoT and other distant edge nodes. Finally, the Cloud is viewed as a unit containing multiple processing cores with high-speed connections between cores.

Given the application model $AG = (T, Ch)$ and a platform model $PG = (P, L)$, a valid mapping is a pair of unique assignments $(m_t : T \to P; m_{Ch} : Ch \to L)$. For each task assigned to a processor, an execution schedule is created. The goal of mapping and scheduling is to minimize two objectives: total execution time (makespan) C_{max} and total number of hops between layers N_{max}. The second objective is introduced in such form because it would not be beneficial to just count the number of cores used, because the is no real downside to using multiple cores inside one edge node or inside the cloud, but when sending data between layers, it must be transferred over internet which can incur costs, and it can also be unreliable or unstable, so

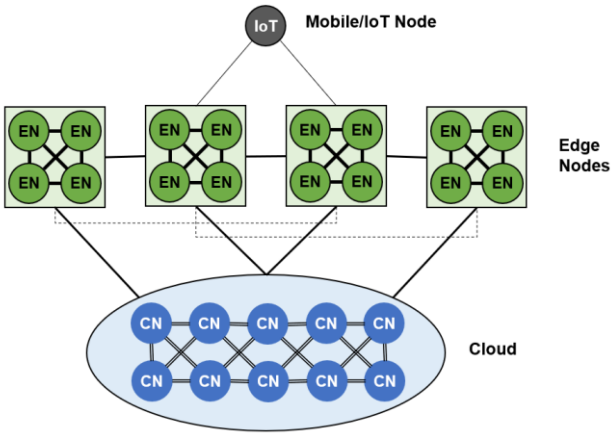


Figure 1 Platform graph

the general idea is to minimize the use of internet.

The multiobjective optimization problem can finally be expressed as

$$min\{C_{max}(x), N_{max}(x)\} \quad (3)$$

where C_{max} and N_{max} are conflicting objective functions $f_i : \mathbb{R}^n \rightarrow \mathbb{R}$ to be minimized simultaneously.

B. *Optimization algorithms*

Four algorithms, SDSE, N-SDSE, C-SDSE and CN-SDSE, were employed to address the previously outlined multi-objective problem, initially introduced in papers [17] and [23]. All these algorithms are based on the NSGA-II evolutionary metaheuristic, with adjustments made to the search process and fitness evaluation methods to suit the specific complexities of task mapping and scheduling in heterogeneous systems. Originally designed to optimize makespan and total elements used, these algorithms were adapted for this study. While makespan remains the primary objective, the secondary objective was adjusted to minimize the number of hops between layers, aligning with the unique requirements of the considered heterogeneous system.

SDSE is an algorithm specifically designed for task mapping and scheduling within heterogeneous multiprocessor embedded systems featuring multiple communication paths between two processors [17]. Moreover, the algorithm's search process is customized to address a common challenge in embedded and IoT systems, where processors cannot execute all types of tasks. It is designed to entirely circumvent mapping tasks to processors that lack the capability to execute them, thus preventing the creation of infeasible solutions and ensuring a more effective convergence of the search process. However, in scenarios where all processors are not fully interconnected, meaning there is no path between certain pairs of processors, the algorithm may not completely avoid such mappings. In such case, it will attempt to prune them from the search space by setting their objectives to infinity.

While the SDSE algorithm treats all infeasible solutions uniformly, making no distinction between cases with only a few instances of consecutive procedures mapped to unconnected processors and those with many such instances, the N-SDSE algorithm [23] introduces a penalty system. This modification aims to create the gradation of "bad" solutions, which is one of the most popular constraint-handling techniques [28][29]. In this modified approach, the objective values of infeasible solutions are no longer infinite. Instead, when it is impossible to find a feasible mapping for a task, a large penalty is added to each objective value. This modification intends to provide solutions with fewer infeasible connections, which may potentially evolve into viable options through various crossovers and/or mutations, a higher chance to advance to the next generation compared to solutions with more substantial penalties and more infeasible mappings.

The C-SDSE and CN-SDSE algorithms [23] introduce a specific modification tailored for sparsely connected clustered platforms. This modification implements a two-stage task mapping approach: initially assigning tasks to clusters and subsequently mapping each task to a processor within the pre-assigned cluster. Cluster detection on the platform occurs in advance, using a straightforward heuristic where any group of processors connected to the same memory is regarded as a cluster. For the study presented in this paper, the definition of clusters extends to include cores within each Edge Node, cores within Cloud, and Mobile/IoT Node with Edge Nodes to which it has connections. Additionally, in the C-SDSE algorithm, all infeasible solutions have their objective values set to infinity, while in the CN-SDSE algorithm, infeasible solutions are penalized in the same way as it is done in the N-SDSE algorithm.

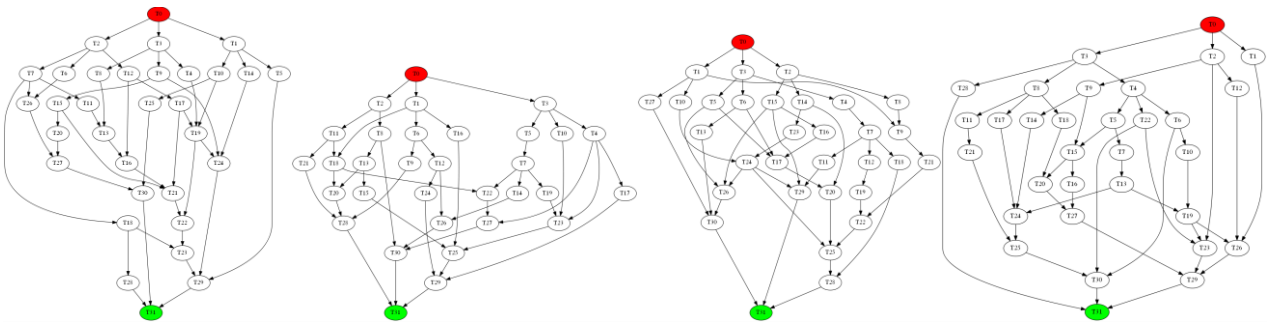
IV. EVALUATION

The performance of the four algorithms was tested on 80 test cases. These cases were designed using four task graphs and four platform models, which were combined and evaluated for each of the five different Communication to Computation Ratios (CCRs), a standard metric used in the synthesis of artificial test cases in which the application is given as DAG [30]. The test cases and the results obtained are presented in the remainder of this section.

A. *System models*

Test cases were built based on four task graphs given in Figure 2, each comprising 30 tasks. The maximum number of concurrent tasks ranges from 7 to 10 in each graph, and loops are not present. The complexity of predecessor-successor connections is the lowest in A1 and the highest in A4.

Furthermore, four platform models P1-P4, all based on the layout depicted in Figure 1, were used. Across all models, three node types are present: Mobile/IoT, Edge, and Cloud. The Mobile/IoT node features a single processing core and is exclusively linked to two Edge nodes. There are a total of four Edge nodes (EN), each equipped with four



A1 A2 A3 A4

Figure 2 Application models

cores (EC). Internally, the connection speed between cores within an Edge node is 100 times faster than that between IoT and Edge. The connection speed between adjacent Edge nodes is four times faster than that between IoT and Edge, and the connection speed between non-adjacent nodes is twice as fast.

Edge nodes are connected to the Cloud, which is represented as a network of ten fully connected cores (CC), each operating 16 times faster than a basic Mobile/IoT node. The connection speed between Edge nodes and the Cloud is 10 times faster than that between Mobile/IoT and Edge, and the connection speed between cores within the cloud is 1000 times faster.

The four models differed in the characteristics of the Mobile/IoT node and the speed of its connection to the Edge layer. For the first model, P1, the Mobile/IoT Node can execute only 50% of tasks from each task graph and is the slowest. In the second model, P2, the Mobile/IoT Node can execute 75% of tasks and is twice as fast as the first model. In the third model, P3, the Mobile/IoT Node can execute all tasks and is four times faster than the first model. In the fourth model, P4, the Mobile/IoT node is eight times faster than the first model, and its connection speed to Edge nodes is two times faster than in previous models. A concise summary of these specifications is presented in Table 1.

TABLE 1 NODE CHARACTERISTICS AND CONNECTION SPEEDS ACROSS FOUR MODELS

Processor type	Execution speed	Connected to : Connection speed
Mobile/IoT1	1	EN : 1
Mobile/IoT2	2	EN : 1
Mobile/IoT3	4	EN : 1
Mobile/IoT4	8	EN : 2
EN	8	IoT : 1 EN (same Edge Node) : 100 EN (adjacent Edge Node) : 4 EN (non-adjacent Edge Node) : 2
CN	16	EN : 10 CN : 100000

For each combination of task graph and platform model, we considered five different CCR values: 0.01, 0.1, 1, 10, and 100.

Regarding the computation of the two objectives C_{max} and N_{max}, the makespan C_{max}, is calculated as the time required to execute all tasks in a taskset for a given mapping. The second objective, N_{max} , quantifies the number of data transfers between distinct nodes, and its value is increased by 1 when data is transferred between IoT and Edge, and by 0.5 when it is transferred between two edge nodes or between edge and cloud.

B. *Results and Discussion*

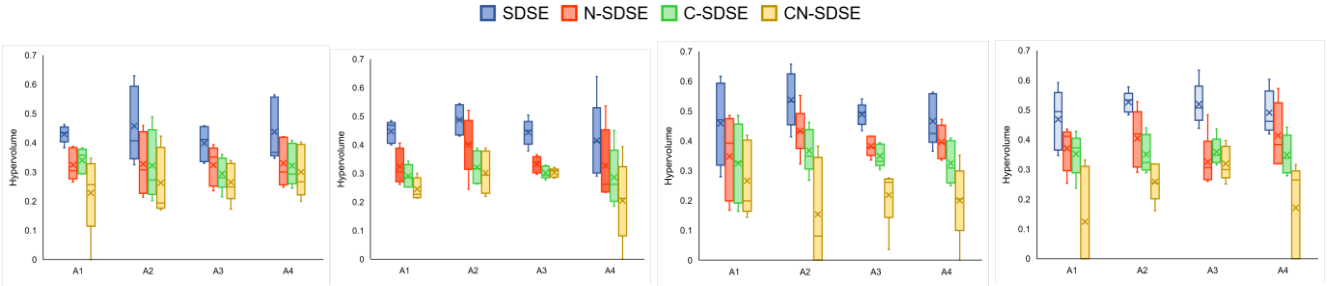
All four algorithms have been tested across all combinations of the task graphs, platform models, and CCRs detailed in the previous section, resulting in a total of 80 test cases.

Basic algorithm parameters such as the population size, stopping criteria, and genetic operators were the same for all algorithms and all test cases, as follows:

- Initial population: 100 randomly generated chromosomes
- Termination criteria: Algorithm halts after 10^5 generations
- Genetic operators employed: 1x, 2x, ux, sbx, spx, pcx, undx, pm, and um
- Each test case underwent 30 repetitions.

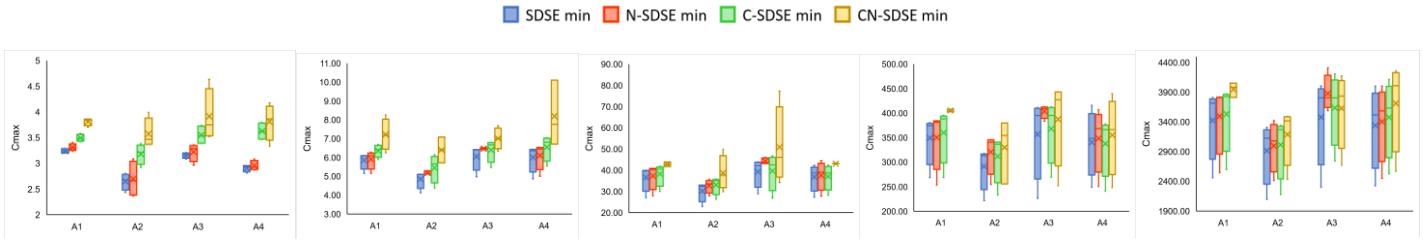
All four algorithms are compared based on the Hypervolume (HV) metric and the values of both objectives C_{max} and N_{max}. The HV metric assesses two critical aspects of an approximation: convergence and diversity [31][32]. Convergence measures the closeness of an approximation to the true Pareto Optimal Front (POF), while diversity reflects the uniformity and extensiveness of the approximation. Calculated as the volume of hyperspace between the algorithm's POF and the nadir point, the worst solution from the unified set of all solutions with a delta added to the extremes for contribution to HV computation. Higher HV indicates superior performance in comparison. For problems with two objectives, HV represents the quadratic surface area between points in the POF and Nadir point.

In Figure 3, the HV metric scores for each platform model (P1-P4) and application model (A1-A4) are presented. The results demonstrate that in each test case the SDSE algorithm consistently outperforms all other algorithms. While C-SDSE and N-SDSE exhibit somewhat similar HV values in most cases, CN-SDSE consistently ranks the lowest. Notably, the CN-SDSE algorithm shows the least success in finding feasible solutions, with instances where no feasible solution was found at all.



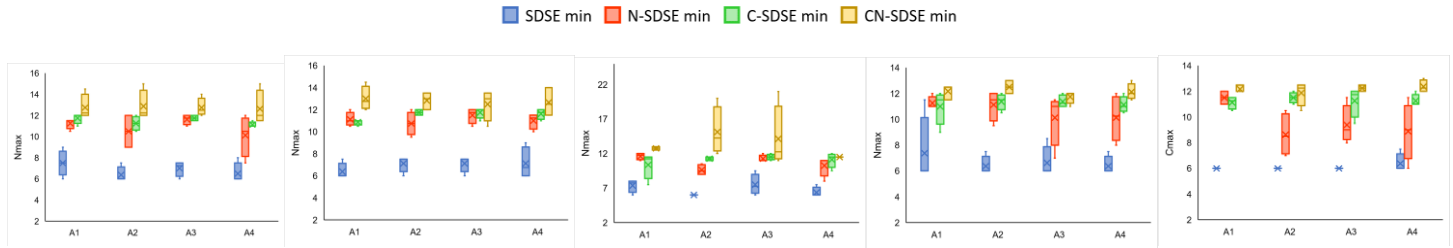
(a) P1 (b) P2 (c) P3 (d) P4

Figure 3 Hypervolume metric scores for platform models P1-P4 and application models A1-A4



(a) CCR=0.01 (b) CCR=0.1 (c) CCR=1 (d) CCR=10 (e) CCR=100

Figure 4 Minimal values of C_{max} objective



a) CCR=0.01 (b) CCR=0.1 (c) CCR=1 (d) CCR=10 (e) CCR=100

Figure 5 Minimal values of N_{max} objective

Figures 4 and 5 depict the range of minimal values of objectives C_{max} and N_{max} identified by each algorithm. The analysis is segmented by each CCR value. Regarding the C_{max} objective, although the SDSE algorithm remains superior overall, the minimal execution times obtained by each algorithm become comparable as CCR increases. Upon closer examination of the values of the C_{max} objective, notably high for CCR values 10 and 100, it becomes evident that all algorithms perform poorly in this case. In such instances, the optimal solution would involve utilizing a maximum of two nodes (provided the IoT/Mobile node cannot execute all tasks), as any other solution would significantly prolong the total makespan due to extensive data exchange between nodes. This suggests that for high CCRs, it may be worth considering further refining the search process, considering only smaller subsets of nodes and prioritizing assigning consecutive tasks to the same processors.

Regarding the N_{max} objective, the SDSE algorithm consistently outperforms others irrespective of CCR changes, with minimal N_{max} values consistently falling within the range of 5-15 across all test cases. This underscores SDSE's superior capability in minimizing traffic between different node types, even in the absence of direct cluster detection support. These findings, coupled with SDSE's strong performance for lower CCRs, suggest that fewer constraints on design space exploration yield better results for this problem. Furthermore, treating all infeasible solutions as equally bad, assigning them an infinite value, proves most effective, as any gradation causes converging around non-optimal points, which provides inferior results. This observation also aligns with previous research on sparsely connected heterogeneous embedded systems.

Conclusion

This paper explored the problem of task mapping and scheduling within a heterogeneous mobile-edge-cloud architecture where nodes have limited connections and constrained task processing capabilities. By adapting four NSGA-II-based algorithms known for handling similar issues in sparsely connected heterogeneous multiprocessor platforms, the study evaluated 80 different scenarios, testing various application and platform models along with different communication-to-computation ratios. The results emphasize the importance of keeping constraints minimal for better outcomes. Additionally, treating all infeasible solutions equally, by giving them an infinite value, proves most effective. Any gradation of infeasible solutions leads to convergence around sub-optimal points, resulting in poorer results. These findings reveal important strategies for improving task mapping and scheduling in complex setups, paving the way for further research and practical implementations.

References

[1] M. Aazam, S. Zeadally, and K. A. Harras, “Fog Computing Architecture, Evaluation, and Future Research Directions,” IEEE Communications Magazine, vol. 56, no. 5, pp. 46–52, May 2018, doi: 10.1109/MCOM.2018.1700707.

[2] J. Pan and J. McElhannon, “Future Edge Cloud and Edge Computing for Internet of Things Applications,” IEEE Internet of Things Journal, vol. 5, no. 1, pp. 439–449, Feb. 2018, doi: 10.1109/JIOT.2017.2767608.

[3] H. Atlam, R. Walters, and G. Wills, “Fog Computing and the Internet of Things: A Review,” Big Data and Cognitive Computing, vol. 2, no. 2, p. 10, Apr. 2018, doi: 10.3390/bdcc2020010.

[4] K. Bilal, O. Khalid, A. Erbad, S.U. Khan, Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers, Comput. Netw. 130 (2018) 94–120, doi: 10.1016/j.comnet.2017.10.002

[5] X. Dai et al., “Task Offloading for Cloud-Assisted Fog Computing With Dynamic Service Caching in Enterprise Management Systems,” IEEE Transactions on Industrial Informatics, vol. 19, no. 1, pp. 662–672, Jan. 2023, doi: 10.1109/TII.2022.3186641.

[6] C. K. Dehury, S. N. Srirama*, P. K. Donta, and S. Dustdar, “Securing Clustered Edge Intelligence With Blockchain,” IEEE Consumer Electronics Magazine, vol. 13, no. 1, pp. 22–29, Jan. 2024, doi: 10.1109/MCE.2022.3164529.

[7] A. Yousefpour et al., “All one needs to know about fog computing and related edge computing paradigms: A complete survey,” Journal of Systems Architecture, vol. 98, pp. 289–330, Sep. 2019, doi: 10.1016/j.sysarc.2019.02.009.

[8] C. Zhao, M. Dong, K. Ota, J. Li, and J. Wu, “Edge-MapReduce-Based Intelligent Information-Centric IoV: Cognitive Route Planning,” IEEE Access, vol. 7, pp. 50549–50560, 2019, doi: 10.1109/ACCESS.2019.2911343.

[9] J. Vergara, J. Botero, and L. Fletscher, “A Comprehensive Survey on Resource Allocation Strategies in Fog/Cloud Environments,” Sensors, vol. 23, no. 9, p. 4413, Apr. 2023, doi: 10.3390/s23094413.

[10] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, “Mobility-Aware Application Scheduling in Fog Computing,” IEEE Cloud Computing, vol. 4, no. 2, pp. 26–35, Mar. 2017, doi: 10.1109/MCC.2017.27.

[11] V. Karagiannis, P. A. Frangoudis, S. Dustdar, and S. Schulte, “Context-Aware Routing in Fog Computing Systems,” IEEE Transactions on Cloud Computing, vol. 11, no. 1, pp. 532–549, Jan. 2023, doi: 10.1109/TCC.2021.3102996.

[12] A. M. Maia, Y. Ghamri-Doudane, D. Vieira, and M. Franklin de Castro, “An improved multi-objective genetic algorithm with heuristic initialization for service placement and load distribution in edge computing,” Computer Networks, vol. 194, p. 108146, Jul. 2021, doi: 10.1016/j.comnet.2021.108146.

[13] S. Iftikhar, M. Golec, D. Chowdhury, S. S. Gill, and S. Uhlig, “FogDLearner: A Deep Learning-based Cardiac Health Diagnosis Framework using Fog Computing,” in Australasian Computer Science Week 2022, New York, NY, USA: ACM, Feb. 2022, pp. 136–144. doi: 10.1145/3511616.3513108.

[14] I. Murturi, A. Egyed, and S. Dustdar, “Utilizing AI Planning on the Edge,” IEEE Internet Computing, vol. 26, no. 2, pp. 28–35, Mar. 2022, doi: 10.1109/MIC.2021.3073434.

[15] S. Iftikhar et al., “AI-based fog and edge computing: A systematic review, taxonomy and future directions,” Internet of Things, vol. 21, p. 100674, Apr. 2023, doi: 10.1016/j.iot.2022.100674.

[16] P. Marwedel, Embedded System Design. Dordrecht: Springer Netherlands, 2011. doi: 10.1007/978-94-007-0257-8.

[17] N. Frid and V. Sruk, “Memory-aware multiobjective design space exploration of heteregeneous MPSoC,” in 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), IEEE, May 2018, pp. 0861–0866. doi: 10.23919/MIPRO.2018.8400159.

[18] M. Raeisi-Varzaneh, O. Dakkak, A. Habbal, and B.-S. Kim, “Resource Scheduling in Edge Computing: Architecture, Taxonomy, Open Issues and Future Research Directions,” IEEE Access, vol. 11, pp. 25329–25350, 2023, doi: 10.1109/ACCESS.2023.3256522.

[19] M. Xu et al., “Genetic Programming for Dynamic Workflow Scheduling in Fog Computing,” IEEE Transactions on Services Computing, vol. 16, no. 4, pp. 2657–2671, Jul. 2023, doi: 10.1109/TSC.2023.3249160.

[20] C. Wang, X. Yu, L. Xu, and W. Wang, “Energy-Efficient Task Scheduling Based on Traffic Mapping in Heterogeneous Mobile-Edge Computing: A Green IoT Perspective,” IEEE Transactions on Green Communications and Networking, vol. 7, no. 2, pp. 972–982, Jun. 2023, doi: 10.1109/TGCN.2022.3186314.

[21] J. Fang and A. Ma, “IoT Application Modules Placement and Dynamic Task Processing in Edge-Cloud Computing,” IEEE Internet of Things Journal, vol. 8, no. 16, pp. 12771–12781, Aug. 2021, doi: 10.1109/JIOT.2020.3007751.

[22] S. Azizi, M. Othman, and H. Khamfroush, “DECO: A Deadline-Aware and Energy-Efficient Algorithm for Task Offloading in Mobile Edge Computing,” IEEE Systems Journal, vol. 17, no. 1, pp. 952–963, Mar. 2023, doi: 10.1109/JSYST.2022.3185011.

[23] N. Frid, V. Sruk, and D. Jakobović, “Design Space Exploration of Clustered Sparsely Connected MPSoC Platforms,” Sensors, vol. 22, no. 20, p. 7803, Oct. 2022, doi: 10.3390/s22207803.

[24] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: NSGA-II,” IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182–197, Apr. 2002, doi: 10.1109/4235.996017

[25] C. Peng, H.-L. Liu, and E. D. Goodman, “Handling multi-objective optimization problems with unbalanced constraints and their effects on evolutionary algorithm performance,” Swarm and Evolutionary Computation, vol. 55, p. 100676, Jun. 2020, doi: 10.1016/j.swevo.2020.100676

[26] T. Adegbija, A. Rogacs, C. Patel, and A. Gordon-Ross, “Microprocessor Optimizations for the Internet of Things: A Survey,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 1, pp. 7–20, Jan. 2018, doi: 10.1109/TCAD.2017.2717782.

[27] Y. Yamato, T. Demizu, H. Noguchi, and M. Kataoka, “Automatic GPU Offloading Technology for Open IoT Environment,” IEEE Internet of Things Journal, vol. 6, no. 2, pp. 2669–2678, Apr. 2019, doi: 10.1109/JIOT.2018.2872545

[28] B. Tessema and G. G. Yen, “An Adaptive Penalty Formulation for Constrained Evolutionary Optimization,” IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, vol. 39, no. 3, pp. 565–578, May 2009, doi: 10.1109/TSMCA.2009.2013333

[29] M. A. Jan and R. A. Khanum, “A study of two penalty-parameterless constraint handling techniques in the framework of MOEA/D,” Applied Soft Computing, vol. 13, no. 1, pp. 128–148, Jan. 2013, doi: 10.1016/j.asoc.2012.07.027.

[30] Y.-K. Kwok and I. Ahmad, “Benchmarking and Comparison of the Task Graph Scheduling Algorithms,” Journal of Parallel and Distributed Computing, vol. 59, no. 3, pp. 381–422, Dec. 1999, doi: 10.1006/jpdc.1999.1578

[31] H.-L. Liu, L. Chen, K. Deb, and E. Goodman, “Investigating the Effect of Imbalance Between Convergence and Diversity in Evolutionary Multi-objective Algorithms,” IEEE Transactions on Evolutionary Computation, pp. 1–1, 2016, doi: 10.1109/TEVC.2016.2606577

[32] H. Ishibuchi, R. Imada, N. Masuyama, and Y. Nojima, “Comparison of Hypervolume, IGD and IGD+ from the Viewpoint of Optimal Distributions of Solutions,” 2019, pp. 332–345, doi: 10.1007/978-3-030-12598-1_27