Computers, Materials & Continua
DOI: 10.32604/cmc.2023.035481
Article

Tech Science Press

Adaptive Resource Planning for AI Workloads with Variable Real-Time Tasks

Sunhwa Annie Nam[1], Kyungwoon Cho[2] and Hyokyung Bahn[3,*]

[1]Department of Computer Science and Engineering, Ewha University, Seoul, 03760, Korea
[2]Embedded Software Research Center, Ewha University, Seoul, 03760, Korea
[3]Department of Computer Science and Engineering, Ewha University, Seoul, 03760, Korea
*Corresponding Author: Hyokyung Bahn. Email: bahn@ewha.ac.kr
Received: 23 August 2022; Accepted: 04 November 2022

Abstract: AI (Artificial Intelligence) workloads are proliferating in modern real-time systems. As the tasks of AI workloads fluctuate over time, resource planning policies used for traditional fixed real-time tasks should be re-examined. In particular, it is difficult to immediately handle changes in real-time tasks without violating the deadline constraints. To cope with this situation, this paper analyzes the task situations of AI workloads and finds the following two observations. First, resource planning for AI workloads is a complicated search problem that requires much time for optimization. Second, although the task set of an AI workload may change over time, the possible combinations of the task sets are known in advance. Based on these observations, this paper proposes a new resource planning scheme for AI workloads that supports the re-planning of resources. Instead of generating resource plans on the fly, the proposed scheme pre-determines resource plans for various combinations of tasks. Thus, in any case, the workload is immediately executed according to the resource plan maintained. Specifically, the proposed scheme maintains an optimized CPU (Central Processing Unit) and memory resource plan using genetic algorithms and applies it as soon as the workload changes. The proposed scheme is implemented in the open-source simulator SimRTS for the validation of its effectiveness. Simulation experiments show that the proposed scheme reduces the energy consumption of CPU and memory by 45.5% on average without deadline misses.

Keywords: Resource planning; artificial intelligence; real-time system; task scheduling; optimization problem; genetic algorithm

1 Introduction

With the recent advances in artificial intelligence (AI) and real-time system technologies, resource planning in traditional hard real-time systems needs to be revisited [1,2]. Specifically, the real-time task set of modern AI workloads is not constant, but the workload varies over time. For example, the task set of autonomous driving changes according to traffic and road situations [3,4]. Similarly, in a smart factory workload, manufacturing processes need to support the re-configurations of production lines

This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

for flexibility [2,5]. Thus, emerging real-time systems for AI workloads should support the ability of adaptive resource planning [1,6]. However, resource plans for traditional real-time systems are determined in advance as they deal with a fixed set of tasks [7].

This paper suggests a new resource planning scheme for a real-time system that schedules a task set consisting of fixed and variable tasks in consideration of the characteristics of emerging AI workloads. A fixed task is a basic task that always exists regardless of the system situation, whereas a variable task can be added temporarily as the system handles a particular situation (e.g., heavy traffic due to accidents in autonomous driving) or a configuration change (e.g., re-configuring a production line in smart factories).

The goal of the proposed planning is to adapt to such workload variations, thereby minimizing the energy consumption of the system and ensuring deadlines for real-time tasks. Since these types of systems operate on limited battery power, saving energy is an important task of resource planning [8,9]. To this end, this paper utilizes CPU (Central Processing Unit) voltage/frequency scaling and memory allocation techniques for saving energy in real-time systems. CPU voltage/frequency scaling provides flexibility between energy-saving and performance by controlling the voltage supplied to the CPU [7,8]. Recently, low-power systems employ some types of low-power memory (also known as non-volatile memory or persistent memory) together with DRAM (Dynamic Random Access Memory) to conserve memory power by assigning tasks to low-power memory rather than DRAM [10].

Even though CPU voltage/frequency scaling and memory allocation techniques save power consumed by CPU and memory respectively, it takes more time to run real-time workloads, which may miss deadlines for some tasks. This is because executing tasks in low CPU voltage/frequency modes and accessing tasks from low-power memory require more time compared to the normal CPU voltage/frequency mode and DRAM allocation [11]. Thus, when planning resources in the system, it is important to determine the CPU and memory configurations that meet deadline constraints as well as minimize energy consumption. To this end, this paper analyzes the CPU voltage/frequency mode and memory allocations for a given task set and optimizes the energy-saving of the system with deadline requirements.

As aforementioned, traditional hard real-time systems deal with a fixed set of tasks whose characteristics (e.g., worst-case execution time and period) are known beforehand, and thus offline resource planning with given task information is widely used [7]. However, as the task set of emerging AI workloads may change as time goes on, the system needs to support the re-planning of resources. However, it is not easy to instantly handle the change of the task set in real-time systems. For example, the resource planner possibly lowers the CPU voltage/frequency mode when there are only a small number of tasks. However, if additional tasks are added at that time, the transition of CPU voltage/frequency modes and new task scheduling is required, which may delay the execution of some real-time tasks.

So, to promptly adapt to workload changes, this paper pre-determines the resource planning for various task combinations consisting of fixed and variable tasks and maintains a set of resource plans. Specifically, the proposed scheme determines the CPU voltage/frequency and memory allocation of all task combinations (i.e., fixed and variable tasks) before executing the workload. Since the optimization of this resource planning is a complicated searching problem known as NP (Non-deterministic Polynomial-time) hard, this paper makes use of the genetic algorithm, which is a well-known optimization technique [12]. That is, as the resource planning should select the CPU voltage/frequency mode and the memory allocation of all tasks that lead to minimizing the energy consumed by CPU and memory, the proposed scheme performs optimizations based on genetic algorithms.

The proposed resource planning consists of two phases. First, finding an optimized resource plan is performed for various workload cases with fixed and variable tasks. It decides the CPU voltage/frequency mode and the memory allocation of the tasks for each case. The system then runs based on the optimized plan for the given workload situation. Second, as the workload changes due to the addition or deletion of some variable tasks, the corresponding resource plan maintained for that situation is adopted. The proposed scheme is implemented in the open-source simulator SimRTS (SIMulator for Real-time Task Scheduling) for the validation of its effectiveness. Simulation experiments under various workload situations show that the proposed resource planning scheme reduces the energy consumed in CPU and memory by 45.5% on average. It is also ensured that the proposed scheme satisfies the deadline requirements of real-time tasks even though the workload is significantly varied over time.

The remainder of this paper is organized as follows. Section 2 explains the proposed workload model for real-time tasks and resource planning. Section 3 describes the details of the genetic optimization procedure for the proposed resource planning scheme. Section 4 presents the simulation result to validate the proposed scheme. Finally, Section 5 concludes this paper.

2 Workload Model and Resource Planning

In the proposed workload model, a fixed task set is defined as $W = \{w_1, w_2, \ldots, w_n\}$, and each task w_i is characterized by $\langle WCET_{w_i}, Period_{w_i}, Size_{w_i}\rangle$, where $WCET_{w_i}$ is the worst-case execution time of w_i, $Period_{w_i}$ is the period of w_i, and $Size_{w_i}$ is the memory size of w_i. For fixed tasks, scheduling can be determined in advance. Meanwhile, to accommodate variable tasks along with fixed tasks, this paper defines a variable task set as $V = \{v_1, v_2, \ldots, v_m\}$, where the task characteristics are defined similarly as those of the fixed task set, i.e., $\langle WCET_{v_i}, Period_{v_i}, Size_{v_i}\rangle$, where $WCET_{v_i}$ is the worst-case execution time of v_i, $Period_{v_i}$ is the period of v_i, and $Size_{v_i}$ is the memory size of v_i. This paper assumes the following for the proposed workload model, which is similar to previous research [13].

A1. Tasks are all independent, implying that a task does not affect others.

A2. The size of DRAM is large enough to accommodate the entire workload of all tasks, but partial hibernation of DRAM is possible if tasks are placed on low-power memory.

A3. The overhead of CPU voltage/frequency scaling and context switch is negligible.

A4. A task can be preempted during its execution.

The proposed scheme takes advantage of CPU voltage/frequency scaling and memory allocation techniques, respectively, to save energy consumed in CPU and memory. Voltage/frequency scaling controls the voltage supplied to the CPU and memory allocation utilizes low-power memory along with DRAM. When such techniques are made use of, the following feasibility test for the fixed task set should be checked first, implying that the worst-case execution time after utilizing the two techniques must satisfy this inequality.

$$\sum_{i=1}^{n} \frac{Resource_{plan}(WCET_{w_i})}{Period_{w_i}} \leq 1 \tag{1}$$

where $Resource_{plan}(WCET_{w_i})$ is the worst-case execution time of task w_i after utilizing the resource planning techniques of voltage/frequency scaling and memory allocation. If (1) is satisfied, a feasible schedule for the given task set through EDF (Earliest Deadline First) scheduling can be found [7,14].

As the workload changes to add some variable task set, resource planning should hold the following test.

$$\sum_{i=1}^{n} \frac{Resource_{plan}(WCET_{w_i})}{Period_{w_i}} + \sum_{j=1}^{m} \frac{Resource_{plan}(WCET_{v_j})}{Period_{v_j}} \le 1 \tag{2}$$

With this constraint, the proposed scheme selects the CPU voltage/frequency mode and the memory allocation for each task utilizing genetic algorithms that aim to optimize CPU and memory energy consumption. Note that this optimization is not performed when the workload runs but is performed in advance with workloads of a variable task set and the result is maintained for the evolution of workloads. This is possible as the task set in execution may change over time, but all possible task sets are known beforehand.

The system starts its execution based on the resource plan for the fixed task set first. If some variable tasks are added to the workload during the execution, the resource plan is changed to a solution that is maintained for corresponding task combinations. As the workload changes again due to the addition or deletion of some variable tasks over time, the resource plan is updated for the new task set. This allows for the minimization of energy consumption without deadline misses. Fig. 1 briefly shows the proposed resource planning scheme.

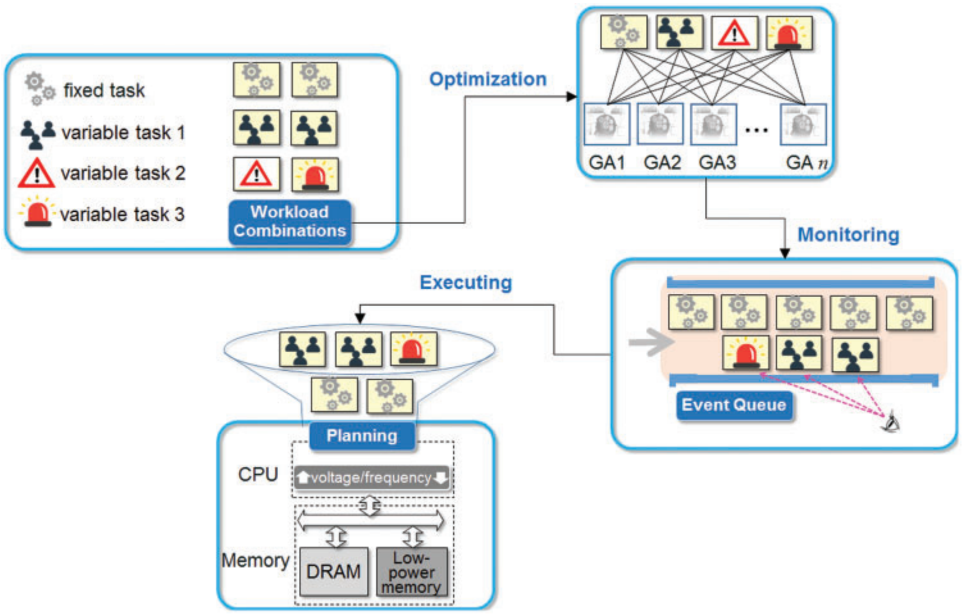


Figure 1: A brief flow of the proposed resource planning scheme

3 Genetic Optimizations for Resource Planning

Genetic algorithms have the ability to search large spaces with limited time budgets to find high-quality solutions similar to population genetics [12]. In the problem of this paper, the search space is the CPU voltage/frequency mode and the memory allocation of all tasks in the workload. The goal of this search is to find a solution that maximally saves the energy of CPU and memory with deadline constraints.

To this end, the proposed scheme evolves a fixed number of solutions repeatedly until finding a sufficiently high-quality solution. Specifically, the genetic algorithm first generates a given number of random initial solutions. Then, the algorithm selects a couple of solutions in the current solution set and crossovers them. Then, a mutation operation is performed for the result of the crossover to get solutions for the next iteration. Finally, the newly generated solution replaces an existing solution in the current solution set. This evolution is repeated until the solution set converges. Then, the algorithm selects the best solution from the final solution set to determine the CPU voltage/frequency modes and the memory allocation of the workload.

3.1 Utility Function

A genetic algorithm makes use of a utility function for measuring the goodness of each solution. As the solution in this paper represents the CPU voltage/frequency mode and the memory allocation, the utility function *Utility*(*i*) of solution *i* is measured based on the energy-saving effect of CPU and memory when the workload runs based on the solution's resource plan. Also, as the optimization problem in this paper has a deadline constraint, a certain penalty value is given in case the resource plan of the solution does not meet the constraint. That is,

$$Utility(i) = Energy\_Saving(i) - Penalty(i) \tag{3}$$

where *Energy_Saving*(*i*) is the saved energy by the solution's resource plan compared to the case the default CPU voltage/frequency and DRAM-only memory are used, and *Penalty*(*i*) is

$$Penalty(i) = \sum_{i=1}^{n} \frac{Resource_{plan}(WCET_{w_i})}{Period_{w_i}} + \sum_{j=1}^{m} \frac{Resource_{plan}(WCET_{v_j})}{Period_{v_j}} - 1 \tag{4}$$

implying the load of tasks that exceeds the limit of possible resource capacity.

3.2 Solution Representations

Each solution in this paper needs to represent CPU and memory resource plans for the given task set. Thus, this paper uses two linear strings: the first string indicates the CPU voltage/frequency mode to run each task, and the second string the memory allocation (i.e., DRAM or low-power memory). There are $n+m$ entries in each string, where n and m are the number of fixed and variable tasks, respectively. Entries in the first string can have a value between 0 and p–1, where p is the number of possible CPU voltage/frequency modes. Similarly, entries in the second string can have the value of 0 or 1 as the proposed scheme makes use of two memory types (i.e., DRAM and low-power memory). The number of solutions the proposed scheme maintains is 100, which is randomly generated when constructing the initial set of solutions.

3.3 Choosing Parent Solutions

For each iteration process, genetic algorithms choose a couple of solutions in the current set, which are called parent solutions. This process is performed probabilistically based on the utility value of the solutions. To evolve the solution set in a better way, the genetic algorithm provides more possibilities for choosing the solution with a higher value of the utility function. To do so, this paper classifies solutions based on their utility value and gives the solutions a selection probability that is linearly proportional to their rank. Specifically, the proposed scheme gives rank 1 a 4x probability compared to rank 100 according to the standard normalization method of the genetic algorithm [12].

3.4 Generating Child Solutions

After choosing two parent solutions, genetic algorithms generate child solutions through two operators, the crossover and the mutation. The crossover operation combines two parent solutions by partially inheriting the characteristics of the two solutions. Among various crossover operators, this paper makes use of the most widely used 1-point crossover. In this operation, the crossover point is chosen randomly within the strings and the child solution is generated by copying the left substring of the crossover point from the parent solution 1 whereas the right substring from the other parent solution 2.

After the crossover, the generated solution is randomly perturbed by the mutation operator. Specifically, mutation changes some entries of the child string with a certain probability. This implies that mutation injects some new features that are not in the current solution set, so it delays the convergence of the solution set. However, this allows genetic algorithms for searching the entire problem space more evenly, thereby preventing the solution set from searching only some limited space.

3.5 Iteration of Evolution

After generating a child solution, a new solution set is formed by replacing a victim solution with the child solution. This paper selects the solution with the least utility value as the victim, which is widely used in steady-state genetic algorithms [12]. The number of iterations is important in genetic algorithms to find a sufficiently good solution. Instead of evolving a constant number of iterations, the proposed scheme monitors whether the solution set converges by checking the utility value of all solutions. That is, this paper performs iterations until all solutions in the solution set have similar utility values. Fig. 2 shows the utility value of the solution set as the number of iterations increases in the proposed genetic algorithm. In particular, the graph plots the utility value of rank 1, rank 100, and the average of all solutions in the solution set over time. As shown in the figure, the utility value of the solution set improves as the number of iterations increases and the solutions finally converge.

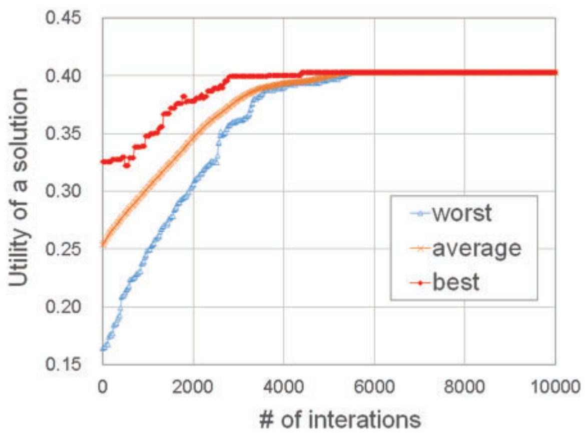


Figure 2: The utility value of the solution set as the number of iterations increases

4 Simulation Results

This section discusses the simulation results performed for validating the proposed resource planning scheme, which is called ARP (Adaptive Resource Planning). For comparison, three other schemes, FRP (Fixed tasks-based Resource Planning), MRP (Maximal tasks-based Resource Planning), and BASELINE are also simulated. BASELINE is a resource configuration that does not adopt either CPU voltage/frequency scaling or low-power memory allocation. That is, BASELINE allocates all tasks to DRAM memory and executes them in a full CPU voltage/frequency mode. FRP and MRP make use of genetic algorithms for optimizing CPU and memory configurations similar to the proposed ARP scheme, but they do not adapt to workload variations. That is, FRP optimizes resource configurations under fixed tasks and does not modify planning while executing the workloads. In contrast, MRP optimizes resource configurations assuming that fixed and variable tasks are maximally executed all the time. The four schemes are implemented in the open-source real-time scheduling simulator SimRTS for performance comparison.

During the simulation, the following four assumptions are made. First, there are four CPU clock frequency modes: 1.0, 0.5, 0.25, and 0.125. Second, the memory capacity of DRAM and low-power memory is configured to load all tasks simultaneously, but the DRAM is partially hibernated when tasks are moved to low-power memory. Third, there is no cost for context switching and CPU mode changes. Fourth, all tasks are independent and can be preempted while they are running. Table 1 summarizes the energy/power consumption and the access latency of DRAM and low-power memory this paper simulates [10,15].

Table 1: Energy/power and access latency characteristics of memory

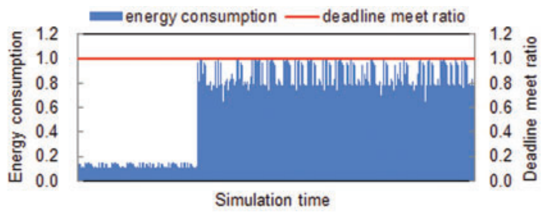
	Low-power memory	DRAM
Write energy	0.1 (nJ/bit)	1.0 (nJ/bit)
Read energy	0.2 (nJ/bit)	0.1 (nJ/bit)
Static power	0.1 (W/GB)	1 (W/GB)
Write latency	350 (ns)	50 (ns)
Read latency	100 (ns)	50 (ns)

The simulations are performed under four workload scenarios to consider a large spectrum of real-time workload cases. For fixed tasks, the CPU load is varied between 0.1 and 0.4. For variable tasks, the CPU load is varied from 0.1 to 0.7. For each scenario, fixed tasks always exist in the system, whereas variable tasks can be added as time progresses. Table 2 summarizes the four scenarios of the simulation. In the table, the sequence of workload means that the task set is activated as time progresses. For example, in Scenario 1, there exist only fixed tasks for the first half of the time, of which the CPU load is 0.2. Then, the variable task set 1 with a CPU load of 0.2 is activated for the next 20% of the total execution time. Note that the fixed tasks are also executed at that time and thus the total CPU load is 0.4. Finally, the variable task set 2 is activated for the last 30% time of the workload. To validate the effectiveness of the proposed scheme in more realistic systems, further experiments are performed in two real workload scenarios: the Robotic Highway Safety Marker (RSM) workload [16] and the Internet-of-Things (IoT) workload [17]. RSM is a real-time task set for the actions of mobile robots carrying safety markers on highways for road construction. IoT is a set of real-time tasks for the actions of controllers in an industrial machine's hands.

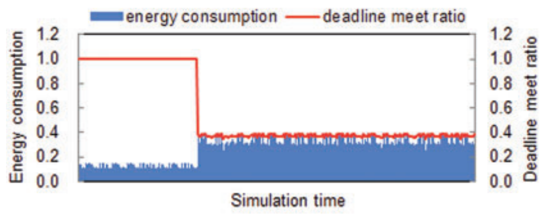
Table 2: Workload scenarios

Scenario	The sequence of workload (CPU load)	Execution time of workloads
1	Fixed (0.2)–variable1 (0.2)–variable2 (0.6)	50%–20%–30%
2	Fixed (0.3)–variable1 (0.5)	30%–70%
3	Fixed (0.1)–variable1 (0.4)–variable2 (0.7)	25%–50%–25%
4	Fixed (0.4)–variable1 (0.1)–variable2 (0.3)	30%–40%–30%

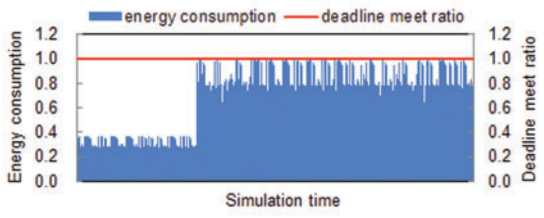
Before discussing the comparison results, this section first shows whether the proposed ARP scheme adapts well to workload fluctuations. Fig. 3 plots the deadline-meet ratio and the energy consumption of Scenario 2 as time progresses. As shown in the figure, the deadline-meet ratio of ARP is not degraded at the time point of 30%, where the workload transition happens. Specifically, the total CPU load of the tasks is changed from 0.3 to 0.8 after 30% of the execution time. This implies that the proposed scheme adapts well to workload evolutions promptly by making use of appropriate resource planning. Also, the energy consumption of ARP increases sharply as the CPU load grows to meet the deadline constraints. On the other hand, FRP and MRP do not adapt well to workload evolutions. In particular, FRP incurs deadline misses when variable tasks are added at the time of 30% and MRP consumes relatively large energy even when the workload is not heavy.



(a) ARP



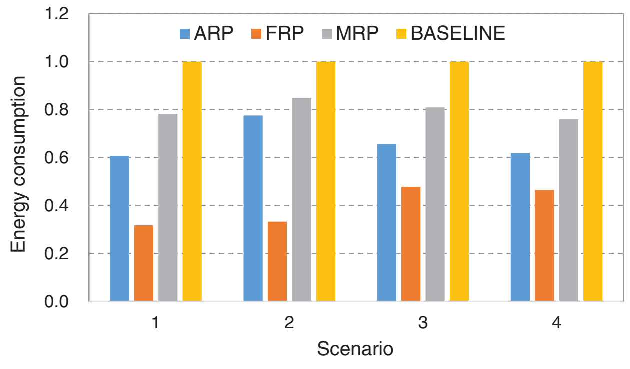
(b) FRP



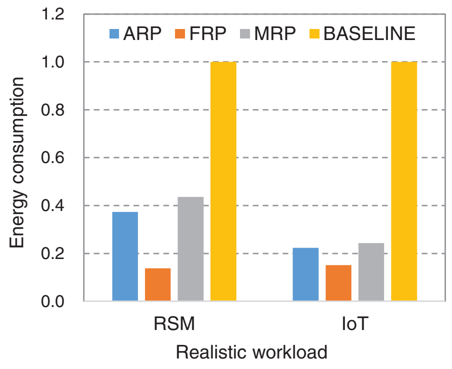
(c) MRP

Figure 3: Adaptation to workload evolutions in ARP

From now on, the four schemes will be compared in terms of energy-saving and deadline-meet ratio. Fig. 4 plots the energy consumption of ARP, FRP, MRP, and BASELINE for the four synthetic and two realistic scenarios. Note that the energy consumption of each scheme is normalized to that of BASELINE. As shown in the figure, ARP reduces energy consumption significantly regardless of any workload scenarios. Specifically, the energy-saving effect of ARP is large in Scenarios 1, 3, and 4 of synthetic workloads and two real workloads where the CPU load is relatively low. This is because there are more chances of adopting low-power techniques of CPU voltage/frequency scaling and memory allocation without missing deadlines when the load is not heavy. By adopting ARP, the energy consumed in CPU and memory is reduced by 45.5% on average compared to BASELINE. FRP saves more energy than ARP especially when the workload changes largely. However, FRP does not satisfy the deadline constraints when variable tasks are activated. MRP does not exhibit good results in terms of energy-saving compared to ARP as the resource planning of MRP assumes the maximum task set during the entire execution time regardless of the activation of variable tasks.



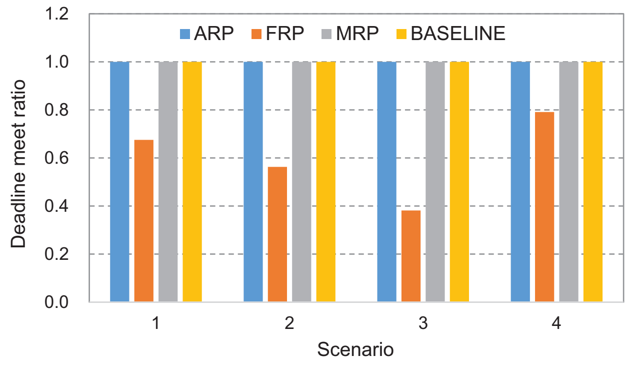
(a) Synthetic workloads



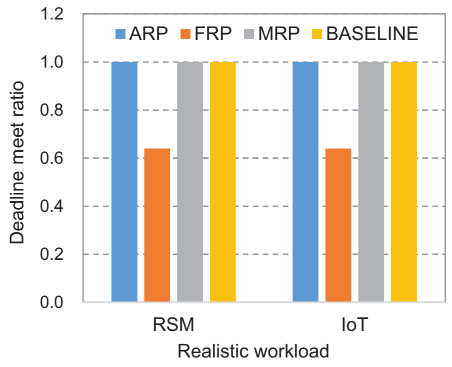
(b) Real workloads

Figure 4: Energy consumption of ARP, FRP, MRP, and BASELINE

Now, the deadline-meet ratio of the four schemes will be discussed. Fig. 5 shows the deadline-meet ratio of ARP in comparison with FRP, MRP, and BASELINE for the four synthetic and two realistic scenarios. As can be seen from this figure, ARP, MRP, and BASELINE do not miss any deadlines at all irrespective of the workload scenarios. Note that ARP saves much more energy than MRP and BASELINE although they show the same results in terms of the deadline-meet ratio. Unlike the three aforementioned schemes, the deadline-meet ratio of FRP is degraded significantly. This is because FRP does not adapt to the change of variable tasks. Specifically, in Scenario 3, where the load of the fixed tasks is very low but the variable tasks generate a large CPU load, the deadline-meet ratio drops the most. The deadline-meet ratio of FRP is 0.68, 0.56, 0.38, and 0.79 for Scenarios 1 to 4, and 0.63 and 0.65 for RSM and IoT workloads, respectively.



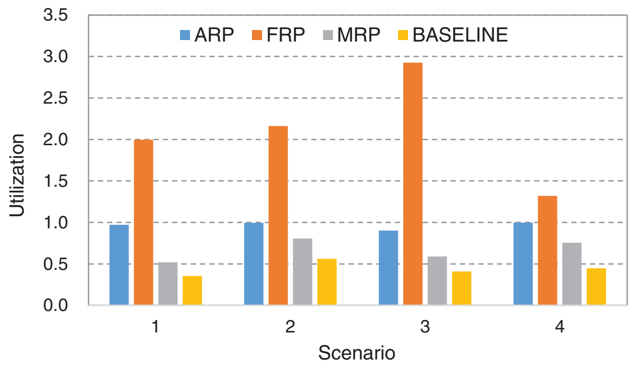
(a) Synthetic workloads



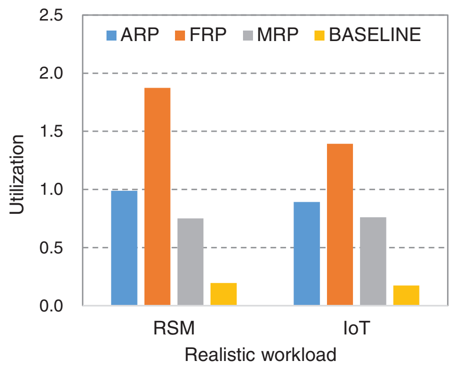
(b) Real workloads

Figure 5: The deadline-meet ratio of ARP, FRP, MRP, and BASELINE

Fig. 6 depicts the resource utilization of the 4 schemes for synthetic and realistic workloads. As can be seen from this figure, ARP shows a high resource utilization of over 0.9 in all cases. Note that the load of the tasks in Scenarios 1 to 4 is always lower than 0.8, implying that ARP applies low-power techniques aggressively to improve the utilization of the given resources, especially for low-load conditions. The utilization of MRP and BASELINE is consistently lower than ARP as the resource planning of these schemes assumes higher load conditions than the actual task set. The utilization of MRP is higher than BASELINE as it makes use of energy-saving techniques but does not react to workload evolutions, limiting the further improvement of resource utilization. The utilization of FRP is too much higher than ARP, but it does not even satisfy the basic deadline requirements as aforementioned in Fig. 5. Note that the utilization should not exceed 1 to properly handle the task set with given resources.



(a) Synthetic workloads



(b) Real workloads

Figure 6: Resource utilization of ARP, FRP, MRP, and BASELINE

In summary, ARP saves energy consumption as much as possible without deadline misses by considering workload evolutions. In particular, when the load of the task set is low, ARP lowers the CPU voltage/frequency modes and allocates more tasks to low-power memory to save more energy. In contrast, when the load of the task set becomes heavy, ARP focuses on real-time constraints by raising the CPU voltage/frequency and the DRAM allocation ratio of tasks.

5 Conclusions

This paper revisited the resource planning issue of real-time systems for AI workloads. To deal with workload changes promptly, the authors proposed a resource planning scheme that classifies real-time tasks into fixed and variable tasks and pre-generates resource plans for any combination of tasks. Specifically, resource planning optimizes CPU voltage/frequency and memory allocation for a given task combination and runs the system based on the plan for the workload situation. Simulation experiments under various workload situations showed that the proposed scheme reduces the energy consumed in CPU and memory by 45.5% on average. It was also confirmed that the proposed scheme meets the deadline requirements of real-time tasks even when the workload changes over time.

Funding Statement: This work was partly supported by the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No.2021-0-02068, Artificial Intelligence Innovation Hub) and (No.RS-2022-00155966, Artificial Intelligence Convergence Innovation Human Resources Development (Ewha University)).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] S. Dehnavi, H. R. Faragardi, M. Kargahi and T. Fahringer, "A reliability-aware resource provisioning scheme for real-time industrial applications in a Fog-integrated smart factory," *Microprocessors and Microsystems*, vol. 70, pp. 1–14, 2019.

[2] J. Wan, M. Yi, D. Li, C. Zhang, S. Wang *et al.,* "Mobile services for customization manufacturing systems: An example of industry 4.0," *IEEE Access*, vol. 4, pp. 8977–8986, 2016.

[3] B. Wu, F. Iandola, P. Jin and K. Keutzer, "SqueezeDet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving," in *Proc. IEEE CVPR Workshops*, Honolulu, HI, USA, pp. 129–137, 2017.

[4] L. Liu, S. Lu, R. Zhong, B. Wu, Y. Yao *et al.,* "Computing systems for autonomous driving: State of the art and challenges," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6469–6486, 2021.

[5] S. Yoo, Y. Jo and H. Bahn, "Integrated scheduling of real-time and interactive tasks for configurable industrial systems," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 1, pp. 631–641, 2022.

[6] P. Tam, S. Math, C. Nam and S. Kim, "Adaptive resource optimized edge federated learning in real-time image sensing classifications," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 14, pp. 10929–10940, 2021.

[7] Y. Lee, Y. Doh and C. Krishna, "EDF scheduling using two-mode voltage clock scaling for hard real-time systems," in *Proc. ACM CASES*, Atlanta, GA, USA, pp. 221–228, 2001.

[8] H. E. Ghor and E. M. Aggoune, "Energy saving EDF scheduling for wireless sensors on variable voltage processors," *International Journal of Advanced Computer Science and Applications*, vol. 5, no. 2, pp. 158–167, 2014.

[9] S. Sennan, S. Ramasubbareddy, A. Nayyar, Y. Nam and M. Abouhawwash, "LOA-RPL: Novel energy-efficient routing protocol for the internet of things using lion optimization algorithm to maximize network lifetime," *Computers, Materials & Continua*, vol. 69, no. 1, pp. 351–371, 2021.

[10] S. Lee, H. Bahn and S. H. Noh, "CLOCK-DWF: A write-history-aware page replacement algorithm for hybrid PCM and DRAM memory architectures," *IEEE Transactions on Computers*, vol. 63, no. 9, pp. 2187–2200, 2014.

[11] D. Kim, E. Lee, S. Ahn and H. Bahn, "Improving the storage performance of smartphones through journaling in non-volatile memory," *IEEE Transactions on Consumer Electronics*, vol. 59, no. 3, pp. 556–561, 2013.

[12] D. Goldberg and J. Holland, "Genetic algorithms and machine learning," *Machine Learning*, vol. 3, no. 2, pp. 95–99, 1988.

[13] H. Bahn and K. Cho, "Evolution-based real-time job scheduling for co-optimizing processor and memory power savings," *IEEE Access*, vol. 8, pp. 152805–152819, 2020.

[14] H. Chetto and M. Chetto, "Some results of the earliest deadline scheduling algorithm," *IEEE Transactions on Software Engineering*, vol. 15, no. 10, pp. 1261–1269, 1989.

[15] E. Lee, H. Bahn, S. Yoo and S. H. Noh, "Empirical study of NVM storage: An operating system's perspective and implications," in *Proc. IEEE MASCOTS Conf.*, Paris, France, pp. 405–410, 2014.

[16] A. Qadi, S. Goddard and S. Farritor, "A dynamic voltage scaling algorithm for sporadic tasks," in *Proc. 24th IEEE Real-Time Systems Symp. (RTSS)*, Cancun, Mexico, pp. 52–62, 2003.

[17] Z. Wang, Y. Liu, Y. Sun, Y. Li, D. Zhang *et al.,* "An energy-efficient heterogeneous dual-core processor for internet of things," in *Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS)*, Lisbon, Portugal, pp. 2301–2304, 2015.