Reinforcement Learning-Based Joint Reliability and Performance Optimization for Hybrid-Cache Computing Servers

Darong Huang, *Graduate Student Member, IEEE*, Ali Pahlevan, Luis Costero, Marina Zapater, *Member, IEEE,* David Atienza, *Fellow, IEEE*

***Abstract*—Computing servers play a key role in the development and process of emerging compute-intensive applications in recent years. However, they need to operate efficiently from an energy perspective viewpoint, while maximizing the performance and lifetime of the hottest server components (i.e., cores and cache). Previous methods focused on either improving energy efficiency by adopting new hybrid-cache architectures including the resistive random-access memory (RRAM) and static random-access memory (SRAM) at the hardware level, or exploring trade-offs between lifetime limitation and performance of multi-core processors under stable workloads conditions. Therefore, no work has so far proposed a co-optimization method with hybrid-cache-based server architectures for real-life dynamic scenarios taking into account scalability, performance, lifetime reliability, and energy efficiency at the same time. In this paper, we first formulate a reliability model for the hybrid-cache architecture to enable precise lifetime reliability management and energy efficiency optimization. We also include the performance and energy overheads of cache switching, and optimize the benefits of hybrid-cache usage for better energy efficiency and performance. Then, we propose a runtime Q-Learning-based reliability management and performance optimization approach for multi-core microprocessors with the hybrid-cache architecture, jointly incorporated with a dynamic preemptive priority queue management method to improve the overall tasks' performance by targeting to respect their end time limits. Experimental results show that our proposed method achieves up to 44% average performance (i.e., tasks execution time) improvement, while maintaining the whole system design lifetime longer than 5 years, when compared to the latest state-of-the-art energy efficiency optimization and reliability management methods for computing servers.**

***Index Terms*—Computing servers, hybrid-cache, reliability management, preemptive queue management, performance optimization, reinforcement learning**

I. Introduction

NOWADAYS the energy efficiency management of computing servers has been brought into focus due to the increase of number and complexity of computing-intensive tasks [1]. In this context, an approximate power breakdown shows that multi-core processors consume over 37% of the total energy consumption of a computing server [2]. Within a processor, besides cores, the last level cache (LLC) also plays a significant role in the system performance, while encompassing over 44% of its total power consumption [3], [4] because of its large capacity and area. Hence, the scalability of LLC size and number of cores in demand for better performance leads to processor aging and, consequently, conducting energy and lifetime reliability management in server infrastructures.

In order to improve the energy efficiency of multi-core processors, non-volatile memory (NVM), which has near-zero leakage power [5], is a promising technology compared to the traditional static random-access memory (SRAM). Indeed, several types of NVM technologies, such as resistive random-access memory (RRAM), spin-transfer-torque magnetic random-access memory (STT-MRAM), and phase-change random-access memory (PCRAM) can be used. These NVM technologies offer the same high speed as SRAM, but with more storage density and less leakage power. An accumulated multiply accelerator has been proposed and fabricated by fully integrating RRAM and CMOS technology together [6]. A CMOS-RRAM neurosynaptic core also has been fabricated and tested in [7]. However, the reliability limitation (or constraint) is still the major problem for RRAM and hinders its mass application [8]. Their limited endurance, and especially their limited write cycles generally prevent their use for high-performance tasks [5], [9]. Hence, a hybrid-cache technology that uses both SRAM and NVM at the architecture level is proposed to address the shortcomings of both SRAM and NVM-based LLC [5], [10]. Fig. 1 shows the target core and uncore (e.g., hybrid LLC) parts in an actual octa-core microprocessor architecture, which has been used as case study in this work. Nevertheless, prior techniques neglect the utilization of software-level reliability management to tackle the trade-offs between energy efficiency and reliability for the hybrid-cache.

From the core reliability management perspective, there are several underlying reliability issues at the hardware level, such as electromigration (EM), stress migration (SM), time-dependent dielectric breakdown (TTDB), thermal cycling (TC), and negative bias temperature instability (NBTI) [11]. Recently, several studies have started to address these issues at the hardware level by controlling the operating temperature, which has a significant impact on the overall system performance [12], [13], [14]. However, the performance degradation becomes intolerable, especially in a dynamic scenario where different types of tasks and workloads need to be

Darong Huang, Ali Pahlevan, Luis Costero, and David Atienza are with the Embedded Systems Laboratory (ESL), École polytechnique fédérale de Lausanne (EPFL), 1015 Lausanne, Switzerland. E-mail: {darong.huang, ali.pahlevan, luis.costerovalero, david.atienza}@epfl.ch.

Marina Zapater was with the Embedded System Laboratory (ESL), École Polytechnique Fédérale de Lausanne (EPFL), 1015 Lausanne, Switzerland. Now, she is with the School of Engineering and Management of Vaud (HEIG-VD), University of Applied Sciences Western Switzerland (HES-SO), 1401 Yverdon-les-Bains, Switzerland. (e-mail: marina.zapater@heig-vd.ch).

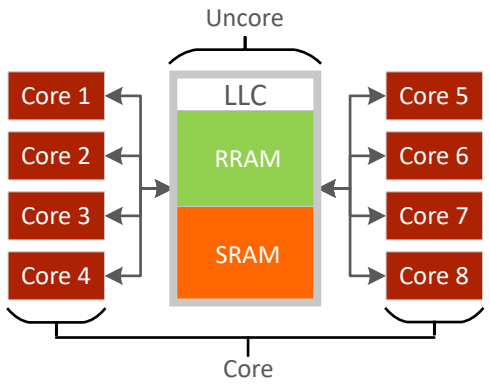


Fig. 1. The Hybrid-cache configuration used in an octa-core microprocessor.

executed. Such a dynamic scenario also increases significantly the overall management complexity. In this context, previous techniques fall short in a joint reliability and task performance management policy for multi-core computing servers.

As complexity raises, finding an optimal solution of overall optimization of the whole system on performance, reliability, and energy efficiency becomes unfeasible at runtime because of non-tolerable computation overhead. Similarly, heuristics are problem-specific and less sensitive to dynamic environments, and their benefits become limited for large and complex problems. Thus, when tackling dynamic problems with large state and/or action spaces, machine learning (ML) methods, and in particular reinforcement learning (RL), are suitable techniques [15], [16], [17]. In real scenarios, reliability management faces the need to incorporate and assess a wide range of metrics (temperature, energy, performance, lifetime, etc.). This challenges the deployment of ML methods due to the large action spaces (e.g., several levels of core frequency, cache selection, etc.). Therefore, the proposal of a runtime method that achieves the highest energy efficiency and task performance, in addition to enhancing system reliability under temperature and reliability restrictions of different components of the processor (i.e., cores and hybrid-cache), remains an open challenge for the computing server processors. Thus, the specific contributions of this work are listed as follows:

- We formulate an RRAM mean-time-to-failure (MTTF) model based on the existing RRAM endurance equation, thus making its runtime lifetime management feasible. Furthermore, we incorporate the cache switching overheads (in energy and performance) to our proposed method to accurately explore the trade-offs between SRAM and RRAM cache technologies' benefits when high-performance applications with different memory access characteristics are executed in real-time.
- We propose a runtime Q-Learning (QL)-based reliability management method to maximize performance of tasks by setting the best cores frequency level, and system energy efficiency by selecting appropriate cache technology for multi-core processors with hybrid-cache architectures (i.e., SRAM and RRAM). A preemptive priority queue management method is also presented to dynamically adjust the priority of tasks execution, and consequently, to improve the overall performance with respect to the tasks' end time constraint.
- Our proposed method achieves up to 44% performance improvement on average and reduces the method computational overhead (i.e., the execution time of the algorithm) by 3x compared to the latest state-of-the-art MPC technique, while ensuring a processor lifetime of 5 years.

The remainder of this paper is organized as follows. Section II reviews related work. In Section III, we provide an overview of the problem description and target scenario. Section IV describes the system modeling used for the multi-core server processor. In Section V, we introduce our proposed QL-based reliability management method. Section VI and VII present the experimental setup and results, followed by the conclusions in Section VIII.

II. Related Work

Our work bring new research contributions in two different areas: 1) energy-efficient hybrid-cache techniques and RRAM endurance modeling, and 2) multicore server reliability management.

A. Energy-Efficient Hybrid-Cache Techniques and RRAM Endurance Modeling

The SRAM technology, although vastly used for LLC in servers, is not suitable for energy efficiency design due to its high leakage power consumption [10]. On the contrary, NVM technologies have higher energy efficiency because of the zero-leakage power characteristic [5]. However, the NVM-based LLC suffers from low endurance cycles and a short lifetime. To fit NVM technology into the high-performance microprocessor, different hybrid-cache architectures (e.g., SRAM+STT-RAM and SRAM+RRAM) are proposed to compensate drawbacks of both SRAM and NVM [10], [18]. These studies target NVM as the main LLC to reduce energy consumption, while SRAM is used for reliability and performance compensation. In this way, the system benefits from both techniques, in terms of performance, reliability, and energy efficiency. Nonetheless, lifetime-aware cache management remains an open challenge.

In order to better quantify the reliability status of NVM, a generalized endurance model is proposed by Strukov [19] to estimate the endurance cycles of NVM. Zhang *et al.* [20] extended Strukov's work and proposed a detailed model to estimate the endurance of RRAM. By applying these modeling works, Beiji *et al.* [21] proposed a novel cache management method to estimate the remaining number of write cycles to the RRAM, and then reduce write accesses to RRAM banks with lower endurance. An endurance and thermal aware management method is also proposed to address RRAM's thermal and reliability issues while processing in-memory applications [22]. However, these approaches are not designed to estimate the lifetime of hybrid-cache architectures and its benefits become limited for high-performance tasks. Furthermore, the existing methods only focus on the reliability aspect of the hybrid-cache architecture, but do not consider the different performance potential of SRAM and RRAM technologies (i.e., fast read accesses to the RRAM and write accesses to the SRAM) to improve the overall system performance.

B. Multicore Server Reliability Management

In this section, we study the related works about server reliability management that aim to maintain the design lifetime of the main components of multi-core microprocessors, including both the core and uncore parts.

1) Core's Reliability Management: Temperature is the main factor that greatly impacts the system's reliability. In this sense, a prevalent reliability management strategy is to throttle the core's frequency and then applying dynamic thermal management techniques [23], [24], [25], [26] to multi-core processors. Finally, the target is to keep the operating temperature below a certain threshold (e.g., 70 °C). However, this strategy severely limits the performance of the server.

In addition, recent studies demonstrate that server failure is induced by different temperature-dependent failure mechanisms, such as EM, SM, TDDB, TC, and NBTI [11]. Based on these models, reliability management methods to address the EM concern are proposed in [27] and [28] by Zhou *et al.* and Wang *et al.*, respectively. Furthermore, another reliability management method is introduced to address both TC and SM problems in [29]. Mercati *et al.* proposed a workload-aware reliability management scheme considering three main reliability issues for Android platform [30]. However, they fall short in covering all of the failure mechanisms for computing server architectures (cf. Section I). In this context, Srinivasan *et al.* [31] have presented a unified reliability model considering all these failure mechanisms for microprocessors that allows to better manage cores' lifetime at runtime. In this direction, [13] proposes a method to deposit lifetime when the temperature is low, then, consume the lifetime deposit for performance boosting during heavy workload periods. Another direction is to increase the lifetime of the cores by throttling the system temperature and, consequently, limiting performance [12]. Nonetheless, existing methods fall short in presenting an efficient method to consider the energy, reliability, and performance trade-offs along with a temperature-dependent lifetime reliability control.

To precisely control cores' reliability, several model predictive control (MPC)-based temperature and reliability management methods have been proposed [32], [33]. However, they suffer from high computational overhead in online management using a fine-grained thermal model. Although recent thermal modeling identification works on multicore systems [34], [35] show that coarse-grained thermal models are effective in predicting temperature and may ease the MPC computational burden, such models have never been tested with MPC controllers for the management of computing servers. Hence, ML methods are promising solutions recently used in the context of reliability management [16]. Nonetheless, these works do not consider the use of comprehensive thermal and reliability control methods for the whole microprocessor (i.e., core and uncore parts) in highly dynamic scenarios.

In order to further improve the performance of computing tasks, different queue management schemes are proposed to be embedded into cores' management schemes. For instance, traditional first come first served (FCFS) approach [36] is widely implemented in existing works [37], [38], [39]. However, FCFS-based methods fail to improve the overall performance of the system due to the fixed execution order of tasks. Thus, shortest job first (SJF) -based methods [40], [41] are implemented to alleviate the congestion problem by bringing the shortest job to the head of the queue. However, SJF-based methods suffer from limited improvements as they put much emphasis on the short execution time tasks but neglect tasks' execution time limit.

2) Uncore's Reliability Management: In a traditional microprocessor architecture, the uncore part (i.e., LLC, memory controller, etc.) is not a concern because of the relatively low temperature and activity ratio compared to the cores. However, with the introduction of the NVM and hybrid-cache techniques, the reliability concern of uncore part is not negligible anymore.

Targeting the proposed hybrid-cache architecture, several reliability management methods [10], [5] have been proposed to maintain NVM's reliability and avoid its failures earlier than the design lifetime. Mittal *et al.* [5] have proposed a method to migrate write-intensive data to SRAM to alleviate the reliability issue of NVM. Chen *et al.* [10] set an access threshold for hybrid-cache design to decide when to power on/off SRAM cache. When the SRAM is powered off, the leakage power consumption is eliminated. However, these methods neglect the NVM's reliability model, which may shorten its lifetime due to setting an inappropriate threshold. In addition, none of the previous works have explored a joint reliability management approach with cores.

III. Problem Description

As explained in Section II, the overall management and optimization of a computing server with hybrid-cache architecture on performance, reliability, and energy efficiency remains an open challenge as it requires a deep assessment of the previous techniques. More importantly, system-wide optimization requires changing the control knobs (e.g., core frequency or operating cache) dynamically during runtime to meet the highly dynamic behaviors of high-performance tasks, and thus to meet the system reliability constraints and optimize the system performance. Therefore, we propose a novel two-level management scheme to incorporate the core and cache reliability management using a runtime QL-based reliability management approach in the first level, while in the second level, a Task Queue Management module is used to minimize the number of overdue tasks (i.e., surpassing the expected end time) by dynamically changing tasks' execution order.

In this section, for better illustration of the proposed methods, we provide a description of the overall scenario, the system we optimize, and the main assumptions taken. Fig. 2 illustrates the proposed scenario and strategy, including key inputs and outputs. The goal of the proposed strategy is to co-optimize the performance, energy consumption, and lifetime reliability of a multi-core server processor equipped with a hybrid-cache architecture (SRAM+RRAM).

From the perspective of the Hybrid-cache, the SRAM component provides higher speed for LLC write accesses but at the cost of high leakage power consumption, while RRAM

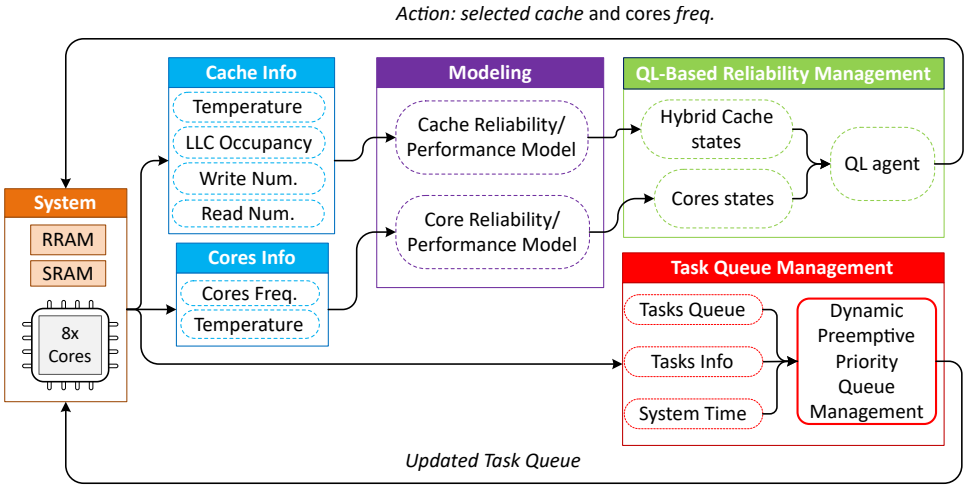


Fig. 2. Overall diagram of the proposed scenario and method, including QL-based reliability management and task queue management blocks.

technology is more energy-efficient and has higher speed in read accesses, but suffers from limited endurance (numbers of write cycles) [42]. Hence, given the system information (LLC occupancy, number of reads and writes to LLC, and cache temperature), and the cache reliability/performance model during the task execution time, our proposal will select the best operating cache for the task to optimize the energy efficiency, reliability, and performance of the system.

From the perspective of the cores' management, our proposal is able to tune the frequency of each core via dynamic voltage and frequency scaling (DVFS) to maximize the task performance while satisfying the cores' lifetime constraints. Similar to modern processors, we assume that each core is able to run at its own frequency.

In the first level of our proposed management approach, we gather these techniques (i.e., adapting cores' frequency and operating cache) in a QL-based reliability management method to achieve the best performance while maintaining the reliability constraints of the system.

For the second level of management, we optimize the management of the task queuing process in multi-core servers, and show how to properly define the optimal order of the execution of the tasks. In our scenario for high-performance computing servers, we assume that multiple (and possibly different) tasks need to be executed on the platform, but only one (occupying all the cores simultaneously) runs on the server at each time. This assumption is legible considering resource managers in high-performance computing scenarios allocate exclusive computing servers for the single coming task [43]. This assumption is also widely adopted in the existing management schemes [44], [1]. In order to improve the overall tasks' performance, the Task Queue Management block takes all the tasks information (i.e., arrival time, amount of time required for being processed on the server - burst time, and expected end time) and tries to re-order the tasks sequence in the queue to reduce the number of overdue tasks (i.e., surpassing expected end time). For this purpose, we present a dynamic preemptive priority queue management method to dynamically suspend, resume, and switch tasks for better performance with respect to their end time limits.

To implement a solution for system-level management to consider all of the above aspects (hybrid-cache, core, and queue management), we need to accurately model the system first. However, state-of-the-art methods [10], [12], [33] do not consider a comprehensive model of the server including cores and the hybrid-cache. In particular, they do not quantify the switching overhead and different performance potentials of SRAM and RRAM technologies. Therefore, in the next section, we introduce a new overall system characterization and its required modeling technique.

IV. System Characterization and Modeling

A. Multi-Core Computing Server Architecture

a) Processor Specification: The target computing server of this work is a Supermicro SuperBlade server, which consists of an octa-core Intel Xeon E5-2667 v4 CPU and 256GB of memory. The CPU is a homogeneous multi-core processor with identical cores, and it has 16 adjustable frequency levels varying from 1.2GHz to 3.5GHz. The cache subsystem comprises a 32KB L1 cache and a 256KB L2 cache. The detailed floorplan of the target microprocessor is illustrated in Fig. 3. Based on this target server, we take real measurements (i.e., CPU load, power trace, LLC occupancy, etc.) for characterizing and modeling the whole system in this work.

b) SRAM and RRAM Hybrid-Cache: A hybrid-cache architecture (SRAM+RRAM) configured with 16MB for each technology and without increasing the LLC area in the floorplan has been used in this work. The characteristics of both technologies are extracted by NVSim simulator [45], as shown in Table I. In the simulator, we tuned the parameters of the general model in accordance with the previous work [46].

B. Cache Switching Overhead Characterization

Due to the LLC data migration, hybrid-cache architectures introduce an overhead when switching the operating cache. This overhead is associated both with energy and performance. The performance overhead (described in Eq. 1) is determined

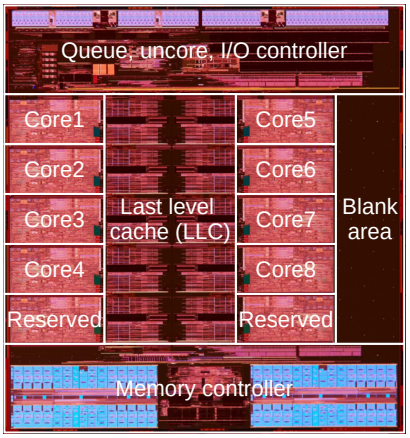


Fig. 3. Floorplan of Intel Xeon E5-2667 v4 CPU.

TABLE I
CHARACTERISTICS OF 16MB SRAM AND RRAM CACHE

	SRAM	RRAM
Area (mm)	11.64	1.37
Read Latency (ns)	5.82	2.71
Write Latency (ns)	3.00	20.93
Read Energy (nJ)	0.58	0.86
Write Energy (nJ)	0.52	0.48
Leakage Power (W)	5.15	0.83
Endurance (Cycles)	10E16	10E6-10E12

by how much data need to be migrated ($size_{data}$) and the migration time for each data block considering both read latency from one cache (t_{read}) and write latency to another cache (t_{write}). The energy overhead (Eq. 2) computation is similar to the performance overhead, but it also considers the leakage power consumption during the whole migration time ($Overhead_{perf}$).

$$Overhead_{perf} = size_{data} \cdot (t_{read} + t_{write}) \tag{1}$$

$$\begin{aligned} Overhead_{energy} = size_{data} \cdot (E_{read} + E_{write}) \\ + P_{leak} \cdot Overhead_{perf} \end{aligned} \tag{2}$$

C. Server Processor Power and Thermal Model

1) Power Characterization: We consider two main contributors to the overall power consumption of the server processor, namely, the core and the uncore power. First, the core power contains a total of 8 cores' power, and each core's power can vary according to the current frequency level and the task being executed. Therefore, we first measure the power consumption profile for a single core, including static and dynamic power, as a function of core's frequency level for a specific task using running average power limit (RAPL) interface [47]. Then, to obtain the total core power, we aggregate all cores' power with respect to their individual frequency level and task information. This is a common methodology adopted from recent works [33], [48].

Second, the uncore power includes all power consumption outside the core region in the processor, such as LLC, memory controller, and I/O subsystems. For traditional SRAM LLC, previous work [48] considers the worst-case power consumption as a constant value. Nevertheless, in this work for hybrid-cache architecture, we consider the leakage power (as shown in Table I) and dynamic power for each technology extracted by the NVSim simulator [45] for a 16MB capacity (the same size for both technologies). We also measure the worst-case memory controller and I/O subsystem power overhead.

2) Processor Thermal Model: The thermal model is used to provide cores and cache temperature information for the reliability model in the simulation framework. It is extracted using the 3D-ICE simulator [49] to characterize our target platform by using the temperature distribution T and power consumption P as follows:

$$T(k) = AT(k-1) + B_d P(k-1) \tag{3}$$

where $T(k-1)$ and $P(k-1)$ are temperature distribution and power consumption of the processor at time $k-1$, respectively. A and B_d are associated with thermal resistance, capacitance, and power injection matrices of the targeted Intel Xeon E5-2667 v4 processor fabricated with $14\,\mathrm{nm}$ technology [50], and are used to compute the temperature for the next time step (i.e., $T(k)$). Finally, $T(k)$ contains temperature distribution of the processor, including cores (T_c) and RRAM cache (T_{RRAM}).

D. Reliability Modelling of Server Processor

We define the lifetime reliability models for the two main components: 1) cores and 2) RRAM cache.

1) Cores Reliability Model: We consider five main failure effects for cores based on previous works [31], [11], [51], [52], including the mean time to failure (MTTF) under the effects of EM (transport of material), SM (caused by mechanical stress), TDDB (caused by gate oxide breakdown), TC (caused by temperature cycling), and NBTI (increasing the transistors' threshold voltage).

Finally, a unified reliability model is formulated using the industry-standard sum-of-failure-rates model [31], as follows:

$$MTTF_{cores} = \left(\sum \frac{1}{MTTF_i} \right)^{-1} \tag{4}$$

where $MTTF_i$ means the aforementioned individual reliability model (i.e., EM, SM, TDDB, TC, and NBTI).

2) RRAM Reliability Model: NVM's reliability models are traditionally defined in terms of endurance, which is measured in terms of the total number of write cycles during the whole lifetime of the cache. A recent study [21] models the endurance of RRAM with the write latency t_w, the activation energy for failure mechanism U_F, the activation energy for the switching mechanism U_S, and device-related constant t_0, as follows:

$$Endurance \propto \left(\frac{t_w}{t_0} \right)^{\frac{U_F}{U_S}-1} \tag{5}$$

According to [21], the write latency coefficient $\frac{t_w}{t_0}$ of RRAM is expressed as:

$$\frac{t_w}{t_0} = \frac{D^2}{t_0 \mu_I (T_{RRAM}) v_w} \left(\frac{r_1 - 1}{2} (x_0^2 - x_f^2) + (r_1 + t_2)(x_f - x_0) \right) \tag{6}$$

where the parameters, except for $\mu_I(T_{RRAM})$, are all material-dependent constants [21]. In addition, $\mu_I(T_{RRAM})$ is given by the following equation [21]:

$$\mu_I(T) = \frac{q_I f a^2 e^{\frac{-E_a}{K_B T_{RRAM}}}}{K_B T_{RRAM}} \tag{7}$$

where the parameters except for RRAM operating temperature T_{RRAM} are also all material-dependent constants [21]. Combining Eq. 7 and Eq. 6, $\frac{t_w}{t_0}$ is expressed in an equation with RRAM's operating temperature T_{RRAM}:

$$\frac{t_w}{t_0} = \frac{D^2 K_B}{t_0 v_w q_I f a^2}\left(\frac{r_1 - 1}{2}(x_0^2 - x_f^2) + (r_1 + t_2)(x_f - x_0)\right) T_{RRAM} e^{\frac{E_a}{K_B T_{RRAM}}} \tag{8}$$

In this work, the parameter c_{r1} is used to simplify the equation by substituting the material constants. Therefore, $\frac{t_w}{t_0}$ is derived as:

$$\frac{t_w}{t_0} = c_{r1} T_{RRAM} e^{\frac{E_a}{K_B T_{RRAM}}} \tag{9}$$

By substituting $\frac{t_w}{t_0}$ in Eq. 5 with Eq. 9, finally we derive the the following Eq. 10 to describe the relationship between RRAM's endurance and its operating temperature. Please note that material-dependent constants, i.e., $c_{r1} = 8.99E-3$ and $c_{r2} = 4.0$, are derived by combining all of the material-dependent parameters used in [21]:

$$Endurance \propto \left(c_{r1} T_{RRAM} e^{\frac{E_a}{k_B T_{RRAM}}}\right)^{c_{r2}-1} \tag{10}$$

where values for other parameters, such as Boltzmann constant k_B and the ion activation energy E_a, are following the recent works in the topic [11], [21].

Based on this endurance model, we formulate an MTTF reliability model for RRAM by dividing endurance with the rate of write to the RRAM (N_{write}), i.e., the lifetime of RRAM is determined by the total amount of writes and write rate (N_{write}) to the RRAM, as follows:

$$MTTF_{RRAM} \propto \frac{\left(c_{r1} T_{RRAM} e^{\frac{E_a}{k_B T_{RRAM}}}\right)^{c_{r2}-1}}{N_{write}} \tag{11}$$

Based on this equation, the MTTF of RRAM ($MTTF_{RRAM}$) decreases with the increase of operating temperature and the number of accesses to the RRAM. Therefore, it is crucial to introduce the temperature-aware runtime reliability management scheme for the RRAM to maintain its reliability.

3) Lifetime Deposit Computation: In contrast to the fixed MTTF threshold used in previous reliability management works [12], [32], a lifetime deposit technique based on lifetime banking [29], [13] is leveraged in this work. Thus, the lifetime deposit (LD) can be expressed as follows:

$$LD = \sum_{t=1}^{k} \left(\frac{1}{MTTF_{nominal}} - \frac{1}{MTTF_t}\right) \tag{12}$$

where $MTTF_{nominal}$ is the nominal MTTF (i.e., 5 years) for the system. If LD is larger than 0, it means that the average of real $MTTF_t$ from time 1 to k (discrete time consistent with Eq. 3) is larger than $MTTF_{nominal}$. Therefore, actual MTTF of the system is guaranteed for not being less than $MTTF_{nominal}$ (i.e., 5 years). Lifetime deposit for cores (LD_{cores}) or RRAM cache (LD_{RRAM}) can be derived by substituting $MTTF_t$ with MTTF information of cores (Eq. 4) or RRAM cache (Eq. 11), respectively.

V. Proposed Q-Learning (QL)-Based Reliability Management Method

As depicted by Fig. 2 in Section III, our proposal consists of two main blocks, namely: 1) QL-based reliability management and, 2) task queue management.

A. Q-Learning (QL)-Based Core and RRAM Reliability Management Method

In this subsection, we introduce how we formulate our new Q-Learning (QL) algorithm [53] to maximize the system performance by selecting the best frequency level per core and operating cache, while guaranteeing lifetime constraints of cores and the hybrid-cache. As illustrated in Fig. 4, the hybrid-cache and cores' information is first sampled from the target system. Then, the runtime information is translated into specific state information for the QL agent. Finally, the QL agent will take relevant actions to finish the control loop by selecting the operating cache and adjusting the cores' frequency.

Typically, RL models are composed of an agent and an environment with a finite set of actions ($\mathcal{A}$) and a state set ($\mathcal{S}$). In the environment, states are observed and actions are applied by the agent (mapping actions to states). Additionally, a reward function (R) is defined for each state-action pair, determining how good is an action to be applied to a specific state. To apply the best action at each moment, a QL agent bases its decisions in an internal q-table storing a q-value associated with each state-action pair. Being the system in a specific state s_t, the final goal of the agent is to apply the action a_t that maximizes the q-value. These q-values are dynamically computed based on multiples observations of the system by means of the Bellman equation, as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{13}$$

where $Q(s_t, a_t)$ is the q-value corresponding to the current state and action pair. R_{t+1} is the reward value after the action a_t is taken, and next state s_{t+1} is observed. α and γ represent the learning rate and discount factor, respectively. In this work, they were set to $\alpha = 0.2$ and $\gamma = 0.2$ to get an optimal learning speed and reduce training overhead. Please note that these values are empirically determined based on the specific application [16], [53]. Learning iteration process ends when the q-table converges.

In our work, at runtime, the agent decides the best frequency levels per core and cache selection ($\mathcal{A}$) considering the runtime system states information ($\mathcal{S}$). After applying the actions, we update the cores' temperature, lifetime deposit states, and remaining instructions for the task, set the server state and

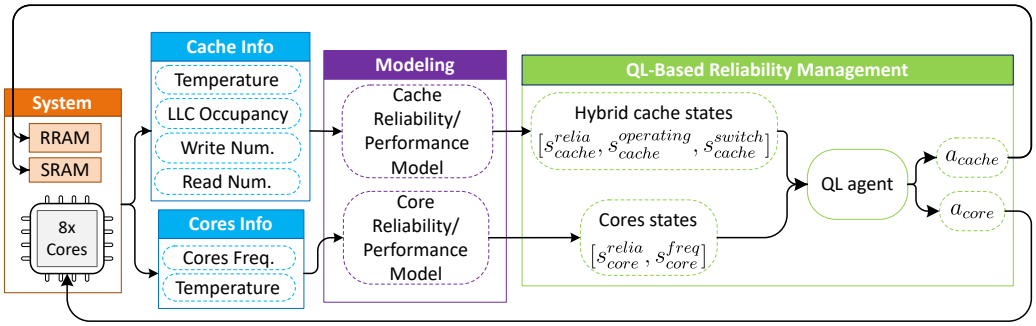


Fig. 4. Workflow of the proposed QL-based reliability management method.

reward, and send them back to the agent. The following subsections describe the specific state, actions, and reward function design for the QL algorithm. Then, the performance and computational overhead of our method are analyzed in Section VII.

1) State Definition: We consider the state definition of the server as a pair of states of the hybrid-cache and cores (i.e., $s = [s_{cache}, s_{core}]$) covering all the important and necessary runtime information for the hybrid-cache and cores.

Regarding the s_{cache}, we take into account three main aspects: temperature-dependent reliability state (s_{cache}^{relia}), current operating cache state ($s_{cache}^{operating}$), and the benefits of cache switching state (s_{cache}^{switch}). Therefore, we define this sub-state as $s_{cache} = \left[s_{cache}^{relia}, s_{cache}^{operating}, s_{cache}^{switch}\right]$.

Then, we define the reliability state s_{cache}^{relia} in terms of the runtime RRAM reliability information (LD_{RRAM}) , which is estimated based on the Eq. 12 introduced in Section IV-D (Lifetime Deposit Computation). We only consider the reliability concerns of RRAM because it has a limited lifetime duration that is at least four orders of magnitude lower than SRAM cache [5]. Typically, SRAM is designed to meet the performance goals and provide a wider working temperature range (i.e., long lifetime period). However, it suffers from high-leakage power and low-energy efficiency. Therefore, we consider three different values for the reliability states for the RRAM cache, s_{cache}^{relia}, defined as follows:

$$s_{cache}^{relia} = \begin{cases} State\ 1: & LD_{RRAM} > \theta_{RRAM} \\ State\ 2: & 0 < LD_{RRAM} \leq \theta_{RRAM} \\ State\ 3: & LD_{RRAM} \leq 0 \end{cases} \tag{14}$$

where θ_{RRAM} is the lifetime deposit threshold for the RRAM cache. We create a safe margin state (i.e., State 2) by setting $\theta_{RRAM} = 15\,\mathrm{s}$ to avoid lifetime deposit of RRAM going directly to negative.

For $s_{cache}^{operating}$, there are a total of two states as only two available operating cache options are considered, namely:

$$s_{cache}^{operating} = \begin{cases} State\ 1: & \text{Current operating cache is RRAM} \\ State\ 2: & \text{Current operating cache is SRAM} \end{cases} \tag{15}$$

Finally, the sub-state s_{cache}^{switch} represents the benefit of switching or keeping the currently operating cache, and is defined as follows:

$$s_{cache}^{switch} = \begin{cases} State\ 1: & benefit_{switch} > benefit_{keep} \\ State\ 2: & benefit_{switch} \leq benefit_{keep} \end{cases} \tag{16}$$

where the benefit of switching ($benefit_{switch}$) and the benefit of keeping currently operating cache unchanged ($benefit_{keep}$) are introduced in Subsection V-A2.

In summary, the state of cache s_{cache} encompasses all the three aspects introduced above, i.e., $s_{cache} = \{(x, y, z) : x \in s_{cache}^{relia}, y \in s_{cache}^{operating}, z \in s_{cache}^{switch}\}$. Considering that s_{cache}^{relia}, $s_{cache}^{operating}$, and s_{cache}^{switch} have 3, 2, and 2 individual elements, respectively, s_{cache} has a total of twelve (i.e., $3 \times 2 \times 2 = 12$) different states to cover all the possible reliability, operating cache, and switch benefits information for the hybrid-cache.

s_{core} is defined in terms of the frequency level and current temperature-dependent lifetime deposit (i.e., LD_{cores}). Each core is able to run at 16 frequency levels, from 1.2GHz to 3.5GHz, therefore, we define the frequency state of the core s_{core}^{freq} to cover all of these 16 frequency states. The 16 frequency levels used in this work influence reliability to different extents. In particular, lower frequency levels increase the lifetime while higher frequency levels consume the lifetime. Therefore, 16 frequency levels have 16 different influences on the lifetime deposit, separating the reliability values into 17 buckets, i.e., 17 reliability states. Consequently, we defined 17 reliability states to encompass all possible reliability effects from 16 frequency levels. In order to make the QL control scheme more robust, we also added 10 more states, with five states below and five states upper the frequency levels' influence range. In the end, the reliability state of the core s_{core}^{relia} has a total of 27 reliability states in this work.

Finally, the overall state $\mathcal{S}$ is designed to integrate all the combinations of s_{cache} and s_{core}, covering all the different states the system can be in our formulation.

2) Estimation of Benefits of Switching and Keeping the Operating Cache: Our proposed flow to estimate the benefit of switching the cache ($benefit_{switch}$) or, alternatively, the benefit of keeping the cache unchanged ($benefit_{keep}$) are detailed in Algorithm 1. This algorithm takes the necessary LLC runtime information as the input (i.e., size of the data in the operating cache, $size_{data}$, predicted read and write accesses to the LLC in the next management step, N_{read} and N_{write}) to compute the performance and energy trade-offs of switching and keeping the operating cache.

The performance and energy overhead of switching cache is firstly computed (lines 1-2), which has already been introduced

Algorithm 1: Estimation of benefits of switching and keeping the operating cache

Input: Size of the data in the operating cache ($size_{data}$), predicted read and write accesses to the LLC in the next management step (N_{read}, N_{write})

Output: benefit of switching and keeping operating cache ($benefit_{swtich}$, $benefit_{keep}$)

1 $Overhead_{perf} = size_{data} \cdot (t_{read} + t_{write})$;
2 $Overhead_{energy} = size_{data} \cdot (E_{read} + E_{write}) + E_{leak} \cdot Overhead_{perf}$;
3 $Overhead_{switch} = w_1 \cdot Overhead_{energy} + w_2 \cdot Overhead_{perf}$;
4 $Overhead_{keep} = 0$;
5 $Gain^{switch}_{perf} = N_{write} \cdot \Delta t^{switch}_{write} + N_{read} \cdot \Delta t^{switch}_{read}$;
6 $Gain^{switch}_{energy} = N_{write} \cdot \Delta E^{switch}_{write} + N_{read} \cdot \Delta E^{switch}_{read} + \Delta E^{switch}_{leakage}$;
7 $Gain_{switch} = w_1 \cdot Gain^{switch}_{energy} + w_2 \cdot Gain^{switch}_{perf}$;
8 $Gain_{keep} = -Gain_{switch}$;
9 $benefit_{swtich} = Gain_{switch} - Overhead_{switch}$;
10 $benefit_{keep} = Gain_{keep} - Overhead_{keep}$;

in Section IV-B. Then, the overall overhead of switching consists of both energy and performance overheads (line 3), where we define the weight factor w_1 for energy and w_2 for performance. In this work, we tried different pairs of w_1 and w_2. Finally, 0.2 and 0.8 were empirically chosen, respectively, to achieve the best trade-off between energy efficiency and performance perspectives. As for other scenarios, w_1 and w_2 can be chosen accordingly to adapt the specific optimization goal, i.e., either strength on energy efficiency by increasing w_1 or strength on performance by increasing w_2. When keeping the operating cache unchanged, there is no overhead (line 4).

Then, the performance and energy gain in the next management step for both switching and keeping cache actions are estimated (lines 5-8). For instance, if the system switched the operating cache, compared with the previous cache type, there is a performance difference between them in both read and write latency. Therefore, we summarize the differences as Δt_{write} and Δt_{read}. Combing with the number of write and read accesses to the cache (N_{write} and N_{read}), the overall performance gain for the next management step can be estimated (line 5). The energy gain of switching the cache can be computed in a similar way (line 6), while it needs to consider the leakage energy difference for the two caches ($\Delta E^{switch}_{leakage}$). Finally, the overall performance and energy gain of cache switch are computed using the same weight factors, i.e., w_1 and w_2, for consistency (line 7). The overall gain for keeping the cache unchanged ($Gain_{keep}$) is the opposite of the $Gain_{switch}$ as they have the opposite operating cache configuration, and consequently, representing opposite performance and energy differences. In the end, the benefits of switching cache and keeping cache unchanged are derived by subtracting overall overhead from the overall gain (lines 9-10).

3) Action Definition: The action set $\mathcal{A}$ includes actions that affects the cache (a_{cache}) and core (a_{core}) independently.

Regarding a_{cache}, only two options exist, namely, either keep using the operating cache ($a_{cache} = 1$) or switching to another cache ($a_{cache} = 2$) to run the task.

We design the action of the core (a_{core}) in terms of changes in the frequency level. Therefore, a_{core} can only get a discrete value in a finite range as $[0, \pm1, \pm2, ..., \pm N_f]$, which means increasing or decreasing the current frequency level by this value (the maximum value of N_f is 16). Selecting the value of N_f affects the QL overhead (execution time) and control quality (converging to the best solution). The higher value of N_f increases the solution quality, but at the expense of higher time and space complexity of the QL algorithm (larger action space). On the contrary, a lower value of N_f provides lower overhead but increases the number of time steps to reach the target state. Therefore, we heuristically choose $N_f = 7$ to balance the QL overhead and control quality.

4) Reward Function: Associated to each pair of state-action, the reward (R) is defined to maximize performance and energy efficiency, while maintaining the cores and cache lifetime limits, as follows:

$$\begin{aligned} R_{core} &= sign(LD_{cores}) \cdot freq \\ R_{cache} &= H(-LD_{RRAM}) \cdot f_1(cache) + H(LD_{RRAM}) \\ &\quad \cdot f_2(a_{cache}) \cdot sign(benefit_{swtich} - benefit_{keep}) \\ R &= \beta_1 \cdot R_{core} + \beta_2 \cdot R_{cache} \end{aligned} \tag{17}$$

where $sign(\cdot)$ is the signum function, which returns 1 when the input value is greater than 0, and -1 otherwise; $H(\cdot)$ is the Heaviside step function, it gives 1 when the input value is greater than 0, and gives 0 for all other scenarios.

When the reliability of cores is guaranteed ($LD_{cores} > 0$), the reward function of cores (R_{core}) would be higher if the frequency of cores ($freq$) increases. This design guarantees that the system reaches its best performance (i.e., highest frequency level) when the reliability of the cores is sustained.

For the cache management, when the reliability of RRAM is low (i.e., $LD_{RRAM} \leq 0$), the reward function (R_{cache}) takes its largest value when SRAM is the operating cache as the function $f_1(cache)$ is designed to return value 1; otherwise, return -1 when RRAM is selected.

If the reliability of RRAM is not the concern (i.e., $LD_{RRAM} > 0$), the second term of the R_{cache} equation will be enabled by the Heaviside function $H(LD_{RRAM})$. The function $f_2(a_{cache})$ gives 1 when the cache switch action is taken ($a_{cache} = 2$); otherwise, it returns -1 when the $a_{cache} = 1$ to continue the usage of current operating cache. The reward function of cache reaches the highest value by switching the operating cache when $benefit_{switch} > benefit_{keep}$, or keeping the operating cache when $benefit_{switch} < benefit_{keep}$. In summary, the cache reward function is designed to maintain the lifetime of RRAM when the reliability is low. Otherwise, it selects the operating cache to optimize the overall energy and performance benefits of the system.

Finally, user-defined weight factors $\beta_i \in [0, 1]$ are used to adjust the importance of each sub-reward function and accordingly compute the overall reward function for the QL agent. Considering that the reliability concerns for both the cores and the hybrid-cache are equally critical for the system

Algorithm 2: Preemptive Priority Queue Management

Input: System time (t_{sys}), Tasks queue ($queue$) and information: burst time (t_{burst}) and end time (t_{end}), instructions left to be executed ($inst_{left}$)
Output: Set task ($task_{exe}$) for execution

1 **Wait until** $mod(t_{sys}, t_{step}) == 0$
2 **if** *queue is empty* **then**
3 System keeps idle;
4 **else**
5 $t^*_{burst} \leftarrow$ Estimate burst time left for each task in the queue based on $inst_{left}$;
6 $task_{urgent} \leftarrow$ Find tasks with $t_{end} - t_{sys} < t^*_{burst}$;
7 **if** $task_{urgent}$ *!= Null* **then**
8 $task_{next} \leftarrow task_{urgent}$;
9 **else**
10 $task_{shortest} \leftarrow$ Find the task with shortest t_{burst};
11 $t_{sys\_next} \leftarrow t_{sys} + task_{shortest}.t^*_{burst}$;
12 $task_{urgent} \leftarrow$ find task with $t_{end} - t_{sys\_next} < t^*_{burst}$;
13 **if** $task_{urgent}$ == *Null* **then**
14 $task_{next} \leftarrow task_{shortest}$;
15 **else**
16 $task_{next} \leftarrow task_{urgent}$;
17 **end**
18 **end**
19 **if** $task_{exe}$ == $task_{next}$ **then**
20 continue executing $task_{exe}$;
21 **else**
22 suspend $task_{exe}$;
23 $task_{exe} \leftarrow task_{next}$;
24 **end**
25 **end**
26 Go to line 1;
27 **end**

operation, we choose 0.5 for both coefficients (i.e., β_1 and β_2) to put the same weight on the cores and hybrid-cache.

B. Dynamic Preemptive Priority Queue Management

Queue management is a key point that greatly impacts overall tasks' performance in a dynamic context. Traditionally, first come first served (FCFS) approach [36] has been used to execute tasks by strictly following the order of tasks arrival time. However, this approach can significantly degrade the overall system performance, especially when a task with a long execution time imposes congestion to the queue.

In order to alleviate drawbacks of existing queue management methods, we consider both a burst time (i.e., processing time on the CPU) and an end time limit of tasks to improve the overall performance. Therefore, based on these factors, we propose a dynamic preemptive priority queue management method, as shown in Algorithm 2.

This algorithm runs in the background and is invoked after reaching a specific time period (t_{step}) (line 1). It is important to note that the system time t_{sys} indicates the reading from the hardware timer. Therefore, it is automatically updated by the system. In the first step, the task queue ($queue$) is checked. If it is empty, the server stays idle (lines 2-3). Otherwise, the algorithm starts to find the next task ($task_{next}$) for execution (lines 5-18). To select $task_{next}$, first, the burst time (t^*_{burst}) of tasks in the $queue$ is estimated based on their number of instructions left ($inst_{left}$) for being executed (line 5). Then, we find urgent tasks ($task_{urgent}$) that breach their end time limits (t_{end}) with respect to the current system time (t_{sys}) and their estimated t^*_{burst} (line 6). If there is more than one urgent task, we consider the earliest arrived task among $task_{urgent}$ as $task_{next}$ (lines 7-8). This rule also applies to the whole algorithm when there is more than one task candidate available. Otherwise, we perform a proactive procedure to select the next task (lines 10-17). In this procedure, we first find the task with the shortest burst time ($task_{shortest}$). Based on the assumption of running this task on the system, we update the system time from t_{sys} to t_{sys_next} (lines 10-11). Then, we check in the urgent task list if the shortest task is being executed (lines 12-13). If the list is empty, we select $task_{shortest}$ as the next task. Otherwise, the new urgent task is selected as $task_{next}$ (lines 13-17). Finally, if the selected task (i.e., $task_{next}$) is the same as current running task ($task_{exe}$), we continue its execution (line 20). Otherwise, $task_{exe}$ is suspended and returned to the queue to resume its execution in the future, and $task_{next}$ is executed (lines 22-23). In the end, the algorithm will go to line 1 and wait until the next iteration. In this regard, in the worst case, the computational complexity of our proposed dynamic preemptive priority queue management method is $\mathcal{O}(n)$.

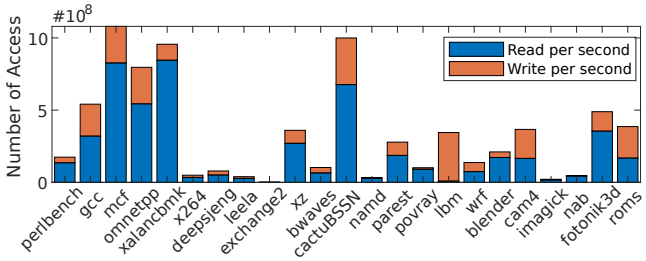


Fig. 5. Different applications' read and write data to the LLC.

VI. Experimental Setup and Comparison Methods

A. Experimental Setup

1) Task Description and Simulation Framework: We use the SPEC CPU 2017 benchmark suite as a set of tasks to simulate realistic scenarios. Benchmarks' power and performance statistics (i.e., power, burst time, instructions, and LLC read/write accesses) under different frequency levels are collected using RAPL and perf tools. Among all the metrics sampled, LLC read/write data for each task is measured with the perf tool with resolution at 1 second to consider the temporal distribution of read/write access. The average LLC read/write data for different applications is shown in Fig. 5 to show the read/write ratio variation among tasks.

We use pqos to sample the runtime LLC occupancy for each task. Fig. 6 shows the LLC occupancy traces for the 4 most representative applications.

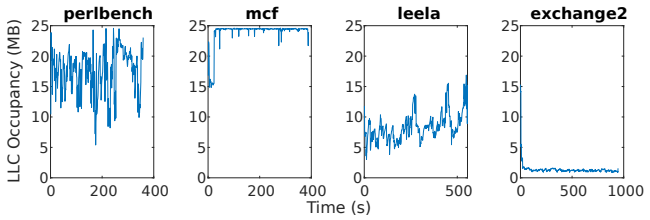


Fig. 6. The LLC occupancy traces for the selected applications.

Given the power, performance, thermal, and reliability models, we develop a high-level simulation framework written in MATLAB to co-optimize the operation of the whole system. The control time step was set to 1 second and all the experiments were performed for a total simulation time of 3×10^7 s (i.e., 347 days) to avoid the possible random effects of different workloads and methods.

In order to evaluate the time overhead of the proposed method, all the comparison methods were implemented and tested on the same system modeled in Section IV, namely, an octa-core Intel Xeon E5-2667 v4 CPU and 256GB of RAM.

2) Experimental Scenario: To mimic the realistic behavior of tasks arriving at different speeds, we use a Poisson distribution with parameter λ to compute the arrival time of tasks, as validated in the literature [1]. Thus, the number of tasks that arrive in the system at each time slot can be tuned by λ. Then, a uniform distribution is used to determine the task type when there is a task coming. More specifically, when a task arrives at the system, a random number from 1 to 23 (a total of 23 SPEC benchmarks) is generated by the uniform distribution to specify the task name.

In general, the load of the server varies during the day, i.e., a heavy workload during the day (large λ), and lower during the night (small λ). Therefore, different λ values are taken into account for such a dynamic scenario, where the system idle ratio varies from 4% to 45% during the simulation time.

B. Comparison Methods

In this work, we compare our proposed method against three different state-of-the-art management methods for computing servers, as follows:

1) Reactive Method (Reactive) [12]: This method sets a fixed reliability threshold at each time (i.e., $MTTF_t > MTTF_{nominal}$) to keep the cores always operating in a safe area by controlling their temperature (i.e., decrease the cores' frequency). This method does not support the hybrid-cache reliability management method, so we assume it always uses SRAM cache for performance and reliability goals.

2) Reactive Method with Hybrid-Cache (Hybrid Reactive) [10]: This method uses an energy-efficient hybrid-cache architecture (i.e., SRAM+NVM), and manages the NVM reliability by setting a threshold for the number of writes to the cache. If the number of write accesses does not exceed the threshold, SRAM is powered off to save energy. Otherwise, SRAM is powered on to reduce the stress of NVM and maintain system reliability. In addition, this method does not support cores reliability management. Therefore, we combine this hybrid-cache management method with the Reactive method [12] to control the whole system.

3) MPC [32], [33]: This method shows the performance benefits of an advanced control policy for core reliability management. The MPC controller will try to reach a higher temperature (i.e., higher operating frequency) if the cores lifetime deposit (LD_{cores}) is still available for high performance consideration. However, hybrid-cache and queue management are important aspects missing from these works. Therefore, we combine the MPC technique with the Hybrid Reactive approach for a fair comparison.

The above three methods do not involve any task queue management policy, therefore, by default, we assume the comparison methods use the FCFS policy [36] for executing the tasks.

VII. Experimental Results

A. Behavior Comparison of Different Server Reliability Management Methods

Figs. 7 to 11 show the cores' temperature, lifetime deposit, cache selection decision, endurance deposit, cache energy consumption, and task executing order for the comparison and proposed methods. We start the evaluation by considering a limited simulation time to better show the key behavior comparison of different methods.

Temperature traces for different methods are shown in Fig. 7. The Reactive method keeps cores' temperature below the preset threshold (i.e., 70 °C) for reliability concerns. This temperature limit guarantees the cores' lifetime longer than the design lifetime (i.e., 5 years) because the lifetime deposit of the Reactive method is always positive, as depicted in Fig. 8(a). However, this fixed conservative temperature threshold is not sensitive to dynamic environments (variable loads of server) and, consequently, degrades the performance. On the contrary, MPC and our proposed QL method achieve a better use of the cores' lifetime deposit to maximize the overall task performance (i.e., increasing cores frequency and operating temperature in Fig. 7(b) and 7(c)) but still maintain the server's design lifetime of 5 years by keeping the lifetime deposit positive according to Fig. 8(b) and 8(c). Thanks to our new fine-grained reliability model, every single core's lifetime deposit can be utilized by both the MPC and QL methods. Therefore, these two methods can tune each core's operating frequency based on their lifetime deposit. More specifically, for the MPC method, temperature traces of tasks 3 and 4 in Fig. 7(b) show how different cores can have different temperature levels (i.e., frequency level). Some cores run out of lifetime deposit and stay at the safe temperature limit (i.e., 70 °C) like the same behavior of Reactive method. On the contrary, the other cores can stay at a higher temperature level until they run out of lifetime deposits. A similar phenomenon can also be found for the QL method in Fig. 7(c).

Due to the lack of hybrid-cache management support, Fig. 9(a) shows how the Reactive method always uses the SRAM cache (red background color), leading to higher leakage power consumption according to Fig. 10(a). As it is shown, the cache energy consumption of the Reactive method is the largest among all the tested approaches and it increases approximately linearly with time because the leakage power

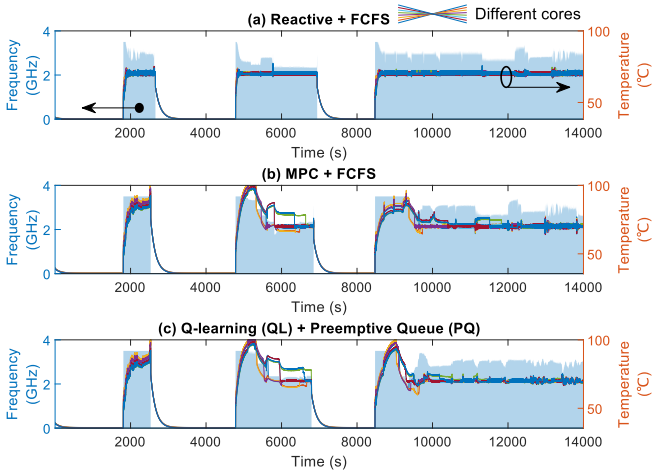


Fig. 7. Temperature comparison of different core management methods with the average frequency as blue background.

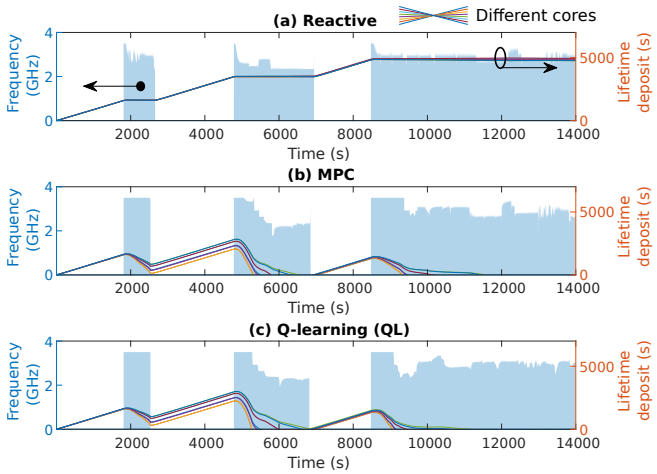


Fig. 8. Cores lifetime deposit comparison of different core management methods with the average frequency as blue background.

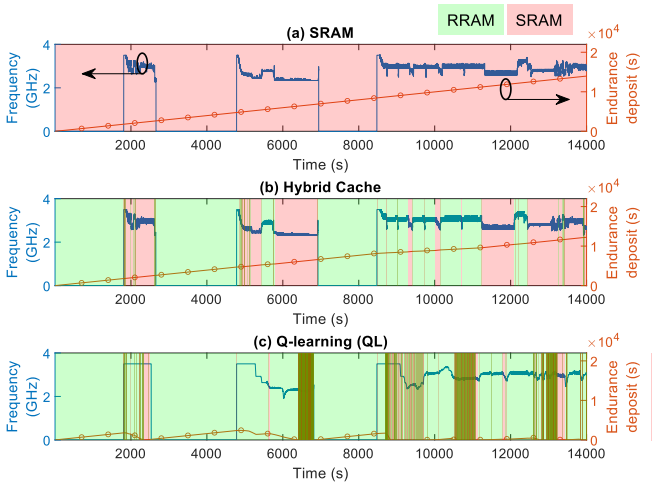


Fig. 9. RRAM lifetime deposit comparison of different methods.

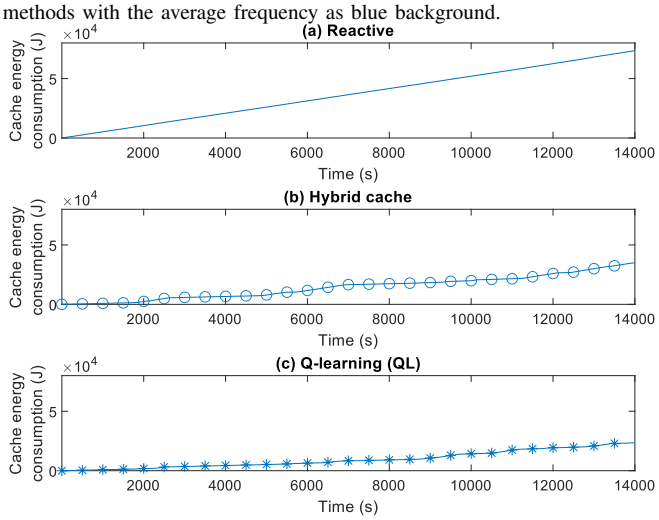


Fig. 10. Cache energy consumption comparison of different methods.

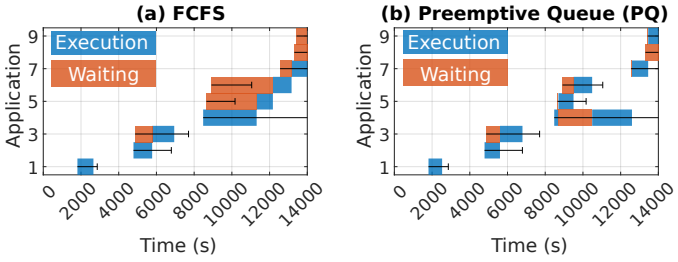


Fig. 11. Queue comparison.

of SRAM takes dominant composition in the overall cache energy consumption. On the contrary, the Hybrid Reactive method sets a write access threshold to decide when SRAM is powered off, and unnecessary leakage power consumption is saved. However, a fixed access threshold cannot optimize the energy efficiency of the hybrid-cache architecture, as it cannot fully exploit the endurance deposit of the hybrid-cache. This can be seen from Fig. 9(b), where the endurance deposit of the Hybrid-Cache method is overabundant at the end of the demo (above 100s). Finally, the proposed QL method exploits RRAM cache more frequently than the other methods for energy efficiency (i.e., lowest cache energy consumption in Fig. 10(c)), while keeping the RRAM lifetime deposit positive (i.e., Fig. 9(c)) to maintain the server's design lifetime.

Fig. 11 demonstrates the outcomes of the different queue management methods, where applications' execution time horizon is represented with blue rectangles and the waiting time horizon is represented with orange rectangles. The solid line in the figure indicates applications' end time limit. Different applications are discerned with different y-axis levels, and the lower y-axis level of the application, the earlier it arrives. The Reactive and MPC approaches use the default FCFS method for task scheduling (i.e., the task arrival order). However, the FCFS method failed to guarantee the end time limit of applications 5 and 6, as shown in Fig. 11(a). In comparison, our proposed queue optimization (Fig. 11(b)) can suspend, reorder, and resume the tasks in the queue (e.g., suspend task 4 when task 5 arrives and reorder task 9 to be executed earlier than task 8) to obtain a better overall performance considering their end time limits and efficient usage of cores and cache lifetime deposits. As explained before in Section VI-A, we have considered the arrival interval of the applications following a Poisson distribution [1], and

TABLE II
RESULTS FOR REACTIVE METHOD (BASELINE)

Total energy	Cache energy	Average PD	Worst -case PD	Overdue tasks (#)	Overdue time (s)
1.4E9	1.7E8	9.6	224.2	6344	1.1E8

therefore, idle periods are present during the experiment. The gap between applications indicates idle time on the system and no executing or queued tasks. Considering a comprehensive comparison is needed for different methods, we take into consideration of energy consumption, temperature, and reliability information during both the active and idle time in this work, and they will be introduced in the following discussions.

B. Energy Efficiency and Performance Comparison for a Long-term Simulation

For a full comparison (covering the whole simulation time of 3×10^7 s), we evaluate the efficiency of different methods (improvements) for different metrics compared to the Reactive method (Table II), as a baseline. The abbreviation "PD" refers to Performance Degradation, which is defined as the task execution time (waiting time + burst time) divided by the shortest burst time the task can achieve on the target system. In this work, we have collected over 16,000 instances of the applications running on the target system, with PD calculated for each instance. The average PD is calculated by taking the mean of these PD values. The worst-case PD is the largest PD value among all of the instances.

Fig. 12 shows the improvements of different methods compared to the baseline (i.e., the Reactive method). The Hybrid Reactive method can reduce 51% cache energy consumption and achieve 10% overall energy savings compared to the Reactive method because of the benefits of hybrid-cache, (i.e., lower energy consumption). Besides a higher energy efficiency, a lower power consumption hybrid-cache means less heat generation, therefore, cores can run at a higher frequency and achieve greater performance. The higher performance also shortens the application's execution time and reduces cores' overall static power consumption. Therefore, the Hybrid Reactive method demonstrates a 6% core energy saving compared to the baseline. In the end, the Hybrid Reactive method achieves good improvements on both average and worst performance degradation, reducing the overdue tasks and time because of the improved system performance compared to the Reactive method. However, it strictly limits cores' temperature at the expense of limited improvements in average PD and many more overdue tasks compared to the MPC method.

The MPC method achieves better improvements than Hybrid Reactive in both performance degradation and overdue task reduction due to removing strict operating temperature threshold for the cores and dynamically managing the temperature and reliability constraints with respect to the server's load. However, running the cores at higher frequencies entails a higher core and total energy consumption, and therefore, the MPC method has the lowest total energy improvements (6%) and consume 1% more core energy than the baseline. In summary, the existing methods bring limited improvements to the computing server with hybrid-cache architecture.

Finally, our proposed QL method improves average and worst-case performance degradation by 37% and 44% compared to the baseline method, respectively. The performance improvements are contributed both by the proposed hybrid-cache management and core management methods. While the QL-based hybrid-cache method selects the operating cache based on the system performance and energy consideration, the QL-based core management method dynamically controls the operating frequency to improve the system performance. In the meantime, the proposed QL method fully exploits the energy efficiency benefits of the hybrid-cache architecture to achieve the best cache, core, and total energy saving (i.e., 63%, 9%, and 15%) among all the comparison methods.

In order to evaluate the improvements of the Dynamic Preemptive Priority Queue Management method (PQ). we formulate the Proposed QL with the PQ and enable the task queue management method. The results show the proposed QL with PQ method have a similar energy consumption with the QL method, but achieve better average performance degradation improvements, and drastically reduce the overdue tasks (i.e., 43% improvements) thanks to the task queue management.

In summary, our proposed approach combines the benefits of a novel QL-based reliability management method and a task queue optimization technique to reach the best overall performance and energy efficiency while ensuring a system lifetime longer than 5 years. Moreover, for other lifetime limit targets, we only need to update $MTTF_{nominal}$ with respect to the desired lifetime duration (i.e., Eq. 12).

C. Computational Overhead (Execution Time) Analysis

In order to achieve a high control accuracy for the MPC method, a complex and accurate thermal model with a large number of thermal cells is needed to be formulated before the implementation of the MPC controller. Besides the pre-thermal modeling work, a thermal model with a large number of thermal cells drastically increases the time and space complexity of the MPC controller, making runtime management unfeasible, as shown in Fig. 13. This is a key reason why the MPC controller has rarely been used in real-life implementations of thermal and reliability control schemes for multi-core microprocessors. On the other hand, the proposed QL method is much faster than MPC (>3x) because QL's time and space overheads are independent of the thermal grid size, thus making it suitable to be applied to large-scale problems and chip sizes, as in latest multi-core microprocessor designs.

VIII. CONCLUSIONS AND FUTURE WORK

In this paper, we have proposed a runtime RL-based reliability management method to optimize the energy efficiency and task performance for computing server processors with a hybrid-cache architecture, while maintaining the system lifetime reliability. First, we have formulated a RRAM MTTF reliability model to meet the RRAM lifetime constraint at runtime. Second, we combined a QL-based reliability-aware

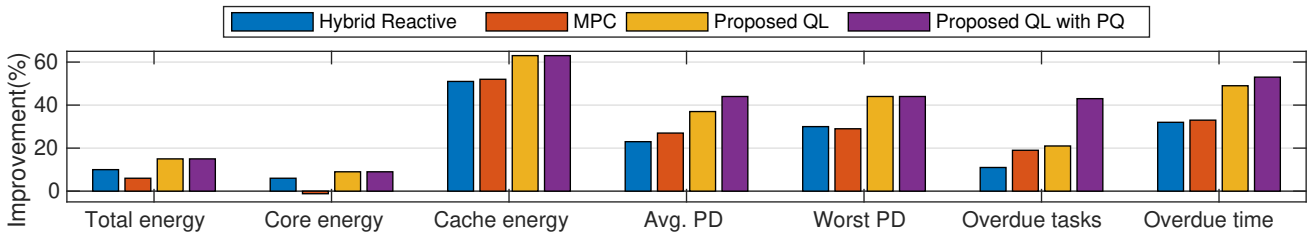


Fig. 12. Different methods improvements compared to baseline.

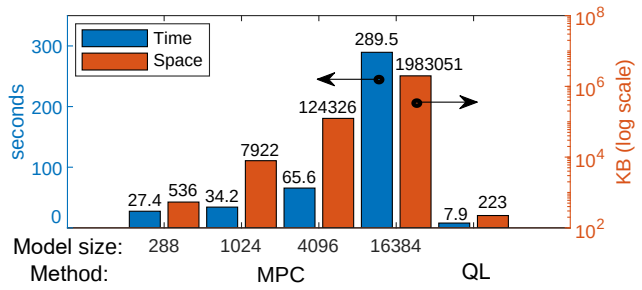


Fig. 13. Computational Overhead comparison of MPC and QL.

management method with a preemptive queue optimization approach to maximize the performance and energy efficiency of the whole system, when running diverse sets of tasks in a dynamic scenario. Our results have shown that the proposed method provides up to 44% performance improvements compared to conventional techniques, while guaranteeing an overall system design lifetime longer than 5 years.

Thanks to the modular design of the proposed method, we can change the reliability, performance, and power models used in this work from RRAM to the accordingly non-volatile memory technology, e.g., STT-MRAM and PCRAM, thus to support other types of hybrid cache technologies. This will be a subject of future work, in which we will study the proposed method's behaviors with other hybrid cache techniques.

Acknowledgment

This work has been partially supported by the EC H2020 RECIPE FET-HPC project (No. 801137), the ERC Consolidator Grant COMPUSAPIEN (No. 725657), and the EC H2020 DeepHealth Project (GA No. 825111).

References

[1] M. Zapater *et al.*, "Leakage-aware cooling management for improving server energy efficiency," *IEEE Trans. on Parallel and Distributed Systems*, vol. 26, pp. 2764–2777, 2015.

[2] C. S. Chan *et al.*, "Optimal performance-aware cooling on enterprise servers," *IEEE TCAD*, vol. 38, no. 9, pp. 1689–1702, 2019.

[3] S. Chakraborty and H. K. Kapoor, "Analysing the role of last level caches in controlling chip temperature," *IEEE Trans. on Sustainable Computing*, vol. 3, no. 4, pp. 289–305, 2018.

[4] A. Pahlevan *et al.*, "Towards near-threshold server processors," in *DATE*, 2016, pp. 7–12.

[5] S. Mittal and J. S. Vetter, "AYUSH: Extending lifetime of SRAM-NVM way-based hybrid caches using wear-leveling," in *IEEE Int. Sym. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2015, pp. 112–121.

[6] W. Wan *et al.*, "33.1 A 74 TMACS/W CMOS-RRAM neurosynaptic core with dynamically reconfigurable dataflow and in-situ transposable weights for probabilistic graphical models," in *IEEE Int. Solid-State Circuits Conference*. IEEE, 2020, pp. 498–500.

[7] F. Cai *et al.*, "A fully integrated reprogrammable memristor–cmos system for efficient multiply–accumulate operations," *Nature Electronics*, vol. 2, no. 7, pp. 290–299, 2019.

[8] W. Banerjee, "Challenges and applications of emerging nonvolatile memory devices," *Electronics*, vol. 9, no. 6, p. 1029, 2020.

[9] Y. Cai *et al.*, "Long live time: Improving lifetime and security for nvm-based training-in-memory systems," *IEEE TCAD*, vol. 39, no. 12, pp. 4707–4720, 2020.

[10] Y.-T. Chen *et al.*, "Dynamically reconfigurable hybrid cache: An energy-efficient last-level cache design," in *DATE*, 2012, pp. 45–50.

[11] JEDEC, "Failure mechanisms and models for semiconductor devices," *JEDEC Publication JEP122H*, 2016.

[12] M. G. Moghaddam *et al.*, "Investigation of DVFS based dynamic reliability management for chip multiprocessors," in *Int. Conf. on High Perf. Comp. & Simulation*, 2015, pp. 563–568.

[13] Z. Lu *et al.*, "Improved thermal management with reliability banking," *IEEE Micro*, vol. 25, no. 6, pp. 40–49, 2005.

[14] A. Iranfar *et al.*, "Thespot: Thermal stress-aware power and temperature management for multiprocessor systems-on-chip," *IEEE TCAD*, vol. 37, no. 8, pp. 1532–1545, 2018.

[15] A. Pahlevan *et al.*, "Integrating heuristic and machine-learning methods for efficient virtual machine allocation in data centers," *IEEE TCAD*, vol. 37, no. 8, pp. 1667–1680, 2017.

[16] A. Iranfar *et al.*, "Machine learning-based quality-aware power and thermal management of multistream HEVC encoding on multicore servers," *IEEE Trans. on Parallel and Distributed Systems*, vol. 29, no. 10, pp. 2268–2281, 2018.

[17] Z. Yang *et al.*, "Enhanced phase-driven q-learning-based DRM for multicore processors," *IEEE TCAD*, vol. 38, no. 11, pp. 2022–2031, 2019.

[18] T.-K. Chien *et al.*, "Write-energy-saving ReRAM-based nonvolatile SRAM with redundant bit-write-aware controller for last-level caches," in *Int. Sym. on Low Power Electronics and Design*, 2017, pp. 1–6.

[19] D. B. Strukov, "Endurance-write-speed tradeoffs in nonvolatile memories," *Applied Physics A*, vol. 122, no. 4, p. 302, 2016.

[20] L. Zhang *et al.*, "Mellow writes: Extending lifetime in resistive memories through selective slow write backs," in *2016 ACM/IEEE 43rd Annual Int. Sym. on Computer Architecture (ISCA)*. IEEE, 2016, pp. 519–531.

[21] M. V. Beigi and G. Memik, "THOR: Thermal-aware optimizations for extending ReRAM lifetime," in *IEEE Int. Parallel & Distributed Processing Sym.*, 2018, pp. 670–679.

[22] M. Zhou *et al.*, "Thermal-aware design and management for search-based in-memory acceleration," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[23] S. A. A. Bukhari *et al.*, "Toward model checking-driven fair comparison of dynamic thermal management techniques under multithreaded workloads," *IEEE TCAD*, vol. 39, no. 8, pp. 1725–1738, 2020.

[24] H. Wang *et al.*, "Leakage-aware predictive thermal management for multicore systems using echo state network," *IEEE TCAD*, vol. 39, no. 7, pp. 1400–1413, 2020.

[25] M. Li *et al.*, "Chip temperature optimization for dark silicon many-core systems," *IEEE TCAD*, vol. 37, no. 5, pp. 941–953, 2018.

[26] J. Saber-Latibari *et al.*, "Ready: Reliability- and deadline-aware power-budgeting for heterogeneous multicore systems," *IEEE TCAD*, vol. 40, no. 4, pp. 646–654, 2021.

[27] J. Zhou *et al.*, "Resource management for improving soft-error and lifetime reliability of real-time mpsocs," *IEEE TCAD*, vol. 38, no. 12, pp. 2215–2228, 2019.

[28] L. Wang *et al.*, "A lifetime reliability-constrained runtime mapping for throughput optimization in many-core systems," *IEEE TCAD*, vol. 38, no. 9, pp. 1771–1784, 2019.

[29] H. Wang *et al.*, "STREAM: Stress and Thermal Aware Reliability Management for 3-D ICs," *IEEE TCAD*, vol. 38, pp. 2058–2071, 2019.

[30] P. Mercati *et al.*, "Warm: Workload-aware reliability management in linux/android," *IEEE TCAD*, vol. 36, no. 9, pp. 1557–1570, 2016.

[31] J. Srinivasan *et al.*, "The case for lifetime reliability-aware microprocessors," in *ISCA*, 2004, pp. 276–287.

[32] F. Zanini *et al.*, "Multicore thermal management with model predictive control," in *DATE*, 2009, pp. 711–714.

[33] H. Wang *et al.*, "Hierarchical dynamic thermal management method for high-performance many-core microprocessors," *ACM TODAES*, vol. 22, no. 1, 2016.

[34] F. Pittino *et al.*, "Robust identification of thermal models for in-production high-performance-computing clusters with machine learning-based data selection," *IEEE TCAD*, vol. 39, no. 10, pp. 2042–2054, 2019.

[35] R. Diversi *et al.*, "Thermal model identification of computing nodes in high-performance computing systems," *IEEE Trans. on Industrial Electronics*, vol. 67, no. 9, pp. 7778–7788, 2019.

[36] Y. Etsion and D. Tsafrir, "A short survey of commercial cluster batch schedulers," School of Computer Science and Engineering, The Hebrew University of Jerusalem, Tech. Rep., 2005.

[37] L. Wang *et al.*, "Towards thermal aware workload scheduling in a data center," in *2009 10th Int. Sym. on pervasive systems, algorithms, and networks*. IEEE, 2009, pp. 116–122.

[38] M. Mohaqeqi *et al.*, "Stochastic thermal control of a multicore real-time system," in *2016 24th Euromicro Int. Conf. on Parallel, Distributed, and Network-Based Processing (PDP)*. IEEE, 2016, pp. 208–215.

[39] S. S. Gill *et al.*, "Thermosim: Deep learning based framework for modeling and simulation of thermal-aware resource management for cloud computing environments," *Journal of Systems and Software*, vol. 166, p. 110596, 2020.

[40] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, *Operating systems: Three easy pieces*. Arpaci-Dusseau Books LLC, 2018.

[41] A. H. El Bakely and H. A. Hefny, "Using shortest job first scheduling in greencloud computing," *Int. Journal of Advanced Research in Computer and Communication Engineering*, vol. 4, pp. 348–354, 2015.

[42] S. Yu and P.-Y. Chen, "Emerging memory technologies: Recent trends and prospects," *IEEE Solid-State Circuits Magazine*, vol. 8, no. 2, pp. 43–56, 2016.

[43] X. Zheng *et al.*, "Exploring plan-based scheduling for large-scale computing systems," in *2016 IEEE Int. Conf. on Cluster Computing (CLUSTER)*. IEEE, 2016, pp. 259–268.

[44] M. A. Salim *et al.*, "Balsam: Near real-time experimental data analysis on supercomputers," *2019 IEEE/ACM 1st Annual Workshop on Large-scale Experiment-in-the-Loop Computing (XLOOP)*, pp. 26–31, 2019.

[45] X. Dong *et al.*, "NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory," *IEEE TCAD*, vol. 31, no. 7, pp. 994–1007, 2012.

[46] C.-H. Jan *et al.*, "A 14nm SoC platform technology featuring 2nd generation tri-gate transistors, 70nm gate pitch, 52nm metal pitch, and 0.0499um2 SRAM cells, optimized for low power, high performance and high density SoC products," in *Sym. on VLSI Tech.*, 2015, pp. 12–13.

[47] J. Pan. RAPL (Running Average Power Limit). [Online]. Available: https://lwn.net/Articles/545745

[48] A. Iranfar *et al.*, "Enhancing two-phase cooling efficiency through thermal-aware workload mapping for power-hungry servers," in *DATE*, 2019, pp. 66–71.

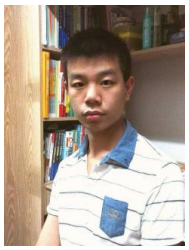
[49] A. Sridhar *et al.*, "3D-ICE: A compact thermal model for early-stage design of liquid-cooled ics," *IEEE Trans. on Computers*, vol. 63, no. 10, pp. 2576–2589, 2014.

[50] Intel Xeon Processor E5-2667 v4. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/92979/intel-xeon-processor-e5-2667-v4-25m-cache-3-20-ghz.html

[51] C. Liu *et al.*, "Service reliability in an HC: Considering from the perspective of scheduling with load-dependent machine reliability," *IEEE Trans. on Reliability*, vol. 68, no. 2, pp. 476–495, 2019.

[52] X. Gao *et al.*, "Investigating security vulnerabilities in a hot data center with reduced cooling redundancy," *IEEE Trans. on Dependable and Secure Computing*, pp. 208–226, 2020.

[53] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1.



Darong Huang is currently a Ph.D. candidate in Electrical Engineering (EE) in the Embedded Systems Laboratory (ESL) at Swiss Federal Institute of Technology Lausanne (EPFL). He received his B.Sc. and M.Sc. degrees in Electrical Engineering from the University of Electronic Science and Technology of China in 2016 and 2019, respectively. His research interests focus on the thermal and reliability management of microprocessors.

Ali Pahlevan is a post-doctoral researcher in the Embedded Systems Laboratory (ESL) at Swiss Federal Institute of Technology Lausanne (EPFL). He received his Ph.D. degree in Electrical Engineering (EE) from EPFL in 2019, M.Sc. degree in Computer Engineering from Sharif University of Technology (SUT) in 2012, and B.Sc. degree in Computer Engineering from Ferdowsi University of Mashhad (FUM) in 2010. He has published over 15 research papers in top international journals and conferences.

Luis Costero Luis M. Costero is a post-doctoral researcher in the Embedded Systems Laboratory (ESL) at Swiss Federal Institute of Technology in Lausanne (EPFL). He received his Ph.D. degree in Computer Engineering from the Universidad Complutense de Madrid (UCM) in 2021. He also obtained a M.Sc. in Computer Engineering from the same university. His main research areas involve high performance computing, asymmetric processors, power consumption and real-time resource management.

Marina Zapater is associate professor in the School of Engineering and Management of Vaud (HEIG-VD) at the University of Applied Sciences Western Switzerland (HES-SO) since 2020. She was a Postdoctoral Research Associate with the Embedded System Laboratory (ESL) at EPFL until February 2020, and is currently an external collaborator of ESL-EPFL. She received her Ph.D. degree in electronic engineering from UPM, Spain, in 2015. Her research interests include thermal, power and performance design and optimization of heterogeneous architectures, from edge devices to high-performance computing processors; and energy efficiency in servers and datacenters. In these fields, she has co-authored more than 75 papers in top-notch conferences and journals. She is an IEEE and CEDA member, and CEDA Assistant VP of finance (2019-2020).

David Atienza (M'05-SM'13-F'16) is professor of electrical and computer engineering, and head of the Embedded Systems Laboratory (ESL) at EPFL, Switzerland. He received his PhD in computer science and engineering from UCM, Spain, and IMEC, Belgium, in 2005. His research interests include system-level design methodologies for high-performance multi-processor system-on-chip and low power Internet-of-Things systems, including thermal-aware design for MPSoCs and many-core servers, and edge AI architectures for wearables and IoT systems. He is a co-author of more than 350 papers, one book and 14 patents in these fields. Among others, Dr. Atienza has received the ICCAD 2020 10-Year Retrospective Most Influential Paper Award, the 2018 DAC Under-40 Innovators Award, an ERC Consolidator Grant in 2016, the 2013 IEEE CEDA Early Career Award, and the 2012 ACM SIGDA Outstanding New Faculty Award. He is an IEEE Fellow, an ACM Distinguished Member, and served as IEEE CEDA President (period 2018-2019).