Runtime Energy Minimization of Distributed Many-Core Systems using Transfer Learning

Dainius Jenkus, Fei Xia, Rishad Shafik, and Alex Yakovlev
School of Engineering, Newcastle University, NE1 7RU, UK
E-mail: {d.jenkus1, fei.xia, rishad.shafik, alex.yakovlev} @newcastle.ac.uk

***Abstract*—The heterogeneity of computing resources continues to permeate into many-core systems making energy-efficiency a challenging objective. Existing rule-based and model-driven methods return sub-optimal energy-efficiency and limited scalability as system complexity increases to the domain of distributed systems. This is exacerbated further by dynamic variations of workloads and quality-of-service (QoS) demands. This work presents a QoS-aware runtime management method for energy minimization using a transfer learning (TL) driven exploration strategy. It enhances standard Q-learning to improve both learning speed and operational optimality (i.e., QoS and energy). The core to our approach is a multi-dimensional knowledge transfer across a task's state-action space. It accelerates the learning of dynamic voltage/frequency scaling (DVFS) control actions for tuning power/performance trade-offs. Firstly, the method identifies and transfers already learned policies between explored and behaviorally similar states referred to as Intra-Task Learning Transfer (ITLT). Secondly, if no similar "expert" states are available, it accelerates exploration at a local state's level through what's known as Intra-State Learning Transfer (ISLT). A comparative evaluation of the approach indicates faster and more balanced exploration. This is shown through energy savings ranging from 7.30% to 18.06%, and improved QoS from 10.43% to 14.3%, when compared to existing exploration strategies. This method is demonstrated under WordPress and TensorFlow workloads on a server cluster.**

I. Introduction

The ability to scale and support dynamic user demands and quality of service (QoS) at minimized energy cost is an important objective in distributed many-core systems [1]. Coarse-grained resource management, such as compute node allocation, is provided by a technique referred to as horizontal scaling [2]. Fine-grained control of intra-node computing resources, such as the allocation of CPU cores, memory shares, or dynamic voltage/frequency scaling (DVFS), is known as vertical scaling. A combination of these techniques provides a means for optimally tuning energy/performance tradeoffs, but generates a complex decision space, which can make achieving energy efficiency challenging [2].

The key to controlling the energy efficiency effectively lies in understanding the conflicting energy/performance tradeoffs [2]. Low-complexity controls in the form of rule-based strategies, which depend on historical data (e.g., CPU utilization [3] or workload characteristics [4]), have been explored by industry [2]. Likewise, model-based techniques using DVFS have also been widely adopted to minimize power in server clusters [5]–[7]. To reduce the computational efforts for solving optimization problems, heuristics are often used at the expense of compromised optimality of controls [6], [7]. In many approaches, such as [5], [7], extensive offline profiling is required to model and exploit power-performance tradeoffs. However, given the complexity of decision space, the scalability of controls can be limited as it is challenging and time-consuming to obtain sufficient modeling data in real distributed system environments.

As an alternative, model-free reinforcement learning (RL), e.g., Q-learning, has been used for learning action policies at runtime with little or no domain- or platform-specific knowledge [3]. In RL, the state-action relationships are learned during the exploration and exploitation phases. The action selection can be random (e.g., the ε-*greedy* rule [8]), employ action probabilities (e.g., *Softmax* [9]) or guided by existing policies [10]. The ability to balance exploration-exploitation using metrics such as time, state counters, or temporal-difference (TD) error [9], plays an important role in managing the quality of controls. The TD-error strategies coupled with a probability-based action selection (i.e., Softmax) [9], [11] have been shown to outperform time-based or state counter-based methods. However, in practice, often sparse feedback from the environment and the requirement to perform exploration for each state and action makes the learning convergence relatively slow [10]. This negatively impacts QoS [12] and energy tradeoffs during the exploratory phases.

Transfer learning (TL), as knowledge transfer from a source domain (expert) to the target (learner) domain, has been used to great success by enhancing initial learning performance under the framework of RL [13], [14]. TL can take different forms of knowledge transfer, e.g., provide advice rules, bias exploration using past policies [10], or directly map over the Q-values and reward functions for the target domain [15]. However, the divergence in knowledge representations (e.g., reward functions and state spaces), limited accessibility of expert knowledge, and sampling cost [13], make it challenging to apply TL in distributed systems.

A direct transfer of Q-values can perform poorly due to subtle variations in the dissimilar task domains or differences in software/hardware architectures. This makes the learning convergence to a good policy worse as it takes time to "unlearn" the part of prior policy [15]. Motivated by the discussed challenges, we apply the principles of TL at an intra-task level to accelerate learning and improve the system's

operational optimality through better QoS and energy savings in the real distributed system environment.

This work presents QoS-aware runtime management for energy minimization. The proposed approach incorporates Q-learning, and enhances it with our new TL-based exploration strategy. It demonstrates two crucial advantages over existing methods: faster and self-sufficient learning and operational optimality with better energy efficiency and QoS. These are achieved by directly addressing the challenges introduced by the conceptual simplicity of model-free learning and the exploration-exploitation dilemma [8], such as slow-learning convergence and sub-optimal action selection during exploration. Core to our approach is multi-dimensional knowledge transfer across a task's state-action space. Our intra-task learning transfer (ITLT) applies TL across the states of a task and our intra-state learning transfer (ISLT) uses TL locally within each state. By keeping each of the dimensions straightforward and well focused, our combined method, intra-state and intra-task transfer Q-learning (ISIT-TQL) substantially improves the learning speed and operational optimality of the system whilst maintaining its scalability. Specifically, the main ***contributions*** of this paper are:

- QoS-aware integrated runtime management of intra-compute node resources using DVFS, i.e., vertical scaling, which utilizes Q-learning accelerated by TL-based exploration to provide elastic controls for minimizing energy.
- TL-based exploration strategy for a multi-dimensional knowledge transfer across a task's state space (ISIT-TQL). It encompasses TL within a local state level and across states.
- Extensive experimental validation and comparative analysis under different workload scenarios.

The remainder of this paper is organized as follows: Section II covers system architecture and its components. Section III presents the proposed runtime management. The experimental evaluation and discussion of the results are explained in Section IV. Finally, Section V concludes the paper.

II. System Architecture

A. Compute Node Cluster

Fig. 1 depicts the system architecture used as a case study in this work. Our runtime acts as a resource controller of cluster nodes. The controller and load balancer is implemented on a hardware separate from cluster nodes. Each node is configured with an Odroid XU4 platform [16], which features the Exynos5422 multi-processor SoC. This SoC is based on the Arm big.LITTLE architecture with four big cores (A15) and four energy-efficient little cores (A7).

B. Load Balancer, Monitor and Workload Generator

The HAProxy (1.8.23) [17] load balancer is used for distributing requests across web application instances across the cluster of compute nodes. It provides metrics including response time, user sessions, the status of nodes and allows dynamically disabling/enabling of nodes at runtime. These metrics are used by our controller for carrying out learning and guiding runtime controls. A workload generator is developed

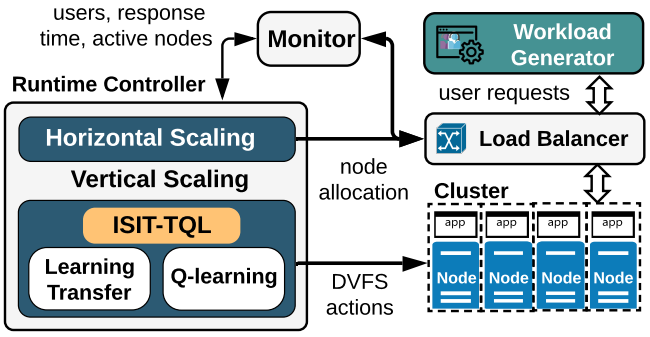

Fig. 1: An overview of the system architecture.

to simulate different workload scenarios using *Python* and is built using the *Wrk 2* load testing tool [18]. It allows to apply workloads with controllable numbers of concurrent users and requests.

III. Proposed Runtime Management

In this section, we first lay down the foundations of Q-learning (Section III-A) and its exploration strategies (Section III-B), then present our novel TL algorithms (Section III-C).

A. Background of Q-Learning

Under the framework of RL [8], Q-Learning is formalized as an off-policy and model-free paradigm. A policy for a set of actions $\mathcal{A}$ and states $\mathcal{S}$, i.e., state-action space ($\mathcal{A} \times \mathcal{S}$) is learned through the use of state-action Q-value function as follows:

$$Q(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma max Q'(s',a') - Q(s,a)], \quad (1)$$

where the learning rate α and the discount factor γ are both in the range of $[0,1]$, the reward R is given for action a in state s at each discrete time step. γ determines the weight of future rewards and scales a Q-value of the new state s'. Finding an optimal policy involves selecting an action at a particular state following two phases: *exploration* and *exploitation* [8].

B. Exploration Strategies

In the next paragraphs, we provide exploration strategies used for comparative analysis.

1) Counter ε-greedy (CE): a well-known ε-greedy exploration rule [8] controls these phases as follows:

$$\pi(s) = \begin{cases} \text{random action from } \mathcal{A}(s), & \text{if } p < \varepsilon(s), \\ \text{argmax}_{a \in \mathcal{A}(s)} Q(s,a), & \text{otherwise,} \end{cases} \quad (2)$$

where $0 \leq p \leq 1$ is a uniform random number generated at each decision step. A random action is chosen (i.e., explored) if $p < \varepsilon(s)$, otherwise the agent selects (i.e., exploits) the best action from the Q-table to maximize the reward. The probability of exploration is controlled with a distinct ε(s) value for each state, estimated as follows:

$$\varepsilon(s) = 1 - e^{-\theta / C(s)}, \quad (3)$$

where θ is a constant to control epsilon (ε) decay rate, and $C(s)$ is the number of times a given state has been visited.

2) ε-greedy-Softmax: a probability-based action selection using *Softmax*, i.e., Boltzmann distribution function, is combined with the ε-greedy exploration rule, which by default implies random action selection. This allows to better balance action selection during the exploration phase than standard ε-greedy and reduce negative short-term rewards [9] as follows:

$$\pi(s) = \begin{cases} \text{Explore: } \frac{e^{Q(s,a)/\tau}}{\sum_{n=1}^{N} e^{Q(s,a_n)/\tau}}, , & \text{if } p < \varepsilon(s), \\ \text{Exploit: } argmax_{a \in \mathcal{A}(s)} Q(s,a), & \text{otherwise,} \end{cases} \quad (4)$$

where $\tau >= 0$ is the temperature parameter and $0 \leq p \leq 1$ is a probability following uniform distribution generated at each decision step. Random actions are explored if $p < \varepsilon$, otherwise the agent exploits the best action. When τ is high, all the probabilities are similar, when it is low, it prioritizes best-rewarded actions, i.e., greedily exploits optimal actions.

3) Adaptive ε-greedy-Softmax (AES): besides knowledge transfer, our complete ISIT-TQL approach utilizes the adaptive control of exploration probability (as defined in [9]). It combines *temporal-difference (TD)* error and *ε-greedy-Softmax* rule as given in Eq. (4). In addition, the method estimates a state dependent exploration probability ε(s) as follows:

$$\varepsilon(s) = \delta \times \frac{1 - e^{\frac{-|\alpha TD_{error}|}{\sigma}}}{1 + e^{\frac{-|\alpha TD_{error}|}{\sigma}}} + (1 - \delta) \times \varepsilon(s), \quad (5)$$

where σ is a positive constant referred as inverse sensitivity, which at low values provides more exploration even for a small change in TD_{error}, whilst at high values gives more weight to exploration when TD_{error} is larger. A constant δ controls the contribution of the given action for exploration, e.g., for the equally-weighted influence of each action, δ is inversely proportional $(1/\mathcal{A})$ to the number of available actions. *Temporal-difference (TD)* error indicates learning divergence from optimal values and is estimated as follows:

$$TD_{error} = R(s,a) + \gamma max Q'(s',a') - Q(s,a). \quad (6)$$

C. Operation of the proposed TL-based Q-learning

Here, we first present essential components of Q-learning (i.e., reward, state, action) related to our case study, then explain the details of our TL-based Q-learning technique.

1) Reward estimation: after a new state (s') and the performance, i.e., response time (t_{perf}) are observed from the monitor, the reward function for a taken action is then estimated as follows:

$$R(s,a) = \begin{cases} \beta_0 \times (T_{slack}/T_{perf}), & \text{if } T_{slack} < 0, \\ \beta_1 \times (1/T_{slack}), & \text{otherwise,} \end{cases} \quad (7)$$

where $T_{slack} = T_{perf} - t_{perf}$, β_1 is a constant for scaling positive rewards, and β_0 scales negative rewards. A negative T_{slack} indicates a QoS violation, while a positive T_{slack} means that the QoS constraint (T_{perf}) is satisfied. The aim is for a low-positive T_{slack}, meaning that the system operates neither too fast (wasting energy) nor too slow (violating the QoS).

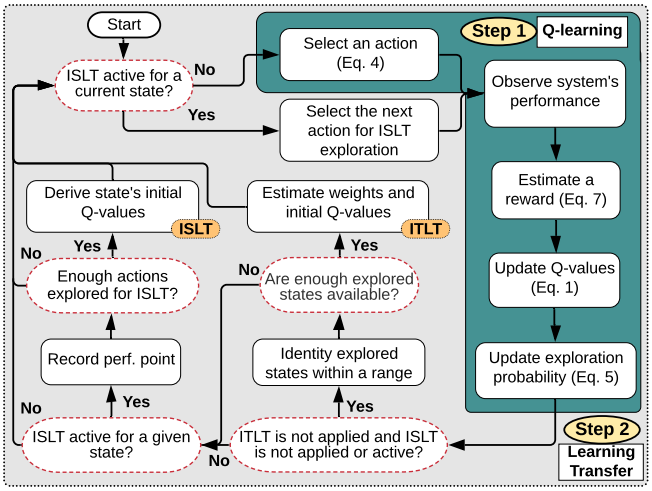

Fig. 2: The workflow of proposed TL-based Q-learning.

2) Definition of state and action: in the multi-process/thread server architectures, a thread-per-connection model is employed to deal with requests from new user connections [19]. This provides scalability but increases the context switching resulting in longer response times as more users connect. The throughput of servers, i.e., requests/sec, is usually much higher than users, and requests per user are often limited at the load balancer. Therefore, rather than individual requests, a *state* is defined as the number of concurrent user connections since it drives the response time.

Control *actions*, as DVFS, are simultaneously applied across enabled nodes to A15 cores in a range from 0.6 to 2.0 GHz with a step of 0.2 GHz. A7 cores run at 1.5 GHz to avoid performance degradation and the lower system energy-efficiency observed when running workloads at lower frequencies.

3) The workflow of ISIT-TQL: Fig. 2 shows the learning workflow of our controller with Q-learning (step 1) and the proposed knowledge transfer (step 2). At each control interval, the method starts by going through a standard Q-learning workflow (step 1). In step 2, exploration progress is assessed and a suitable TL method is applied or initiated (Fig. 2). Step 2 begins by examining conditions for applying ITLT, i.e., identifying explored states $\mathcal{M}_\varepsilon$ within $\mathcal{S}_{range}$. If enough explored states are available, weighted Q-values are estimated as later described in Algorithm 1. If not, ISLT is considered due to the limited availability of explored states. ISTL guides exploration and records performance at selected exploration points. Once ISLT exploration is completed, initial Q-values are derived as later explained in Algorithm 2.

4) Intra-Task Learning Transfer (ITLT): ITLT is defined in Algorithm 1. It uses TD-error based exploration probability $\varepsilon(s)$ (see Eq. (5)) for identifying explored states $\mathcal{M}_\varepsilon$. Both lower (s_l) and higher (s_h) states in reference to the current state (s) are considered within $\mathcal{S}_{dst}$ distance, which is a positive integer (see Fig. 3). A state is explored when $\varepsilon(s) < \mathcal{C}_\varepsilon$, where $\mathcal{C}_\varepsilon \in [0,1]$ is a threshold of exploration (line 2).

After explored states are identified, the $\mathcal{M}_{cond}$ condition is evaluated requiring both lower and higher S_{min} states within

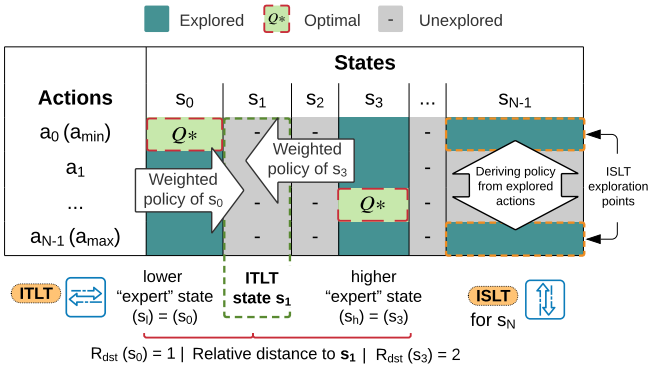

Fig. 3: An overview of multi-dimensional knowledge transfer within state-action space (Q-table) using ISLT and ITLT methods.

Algorithm 1: Intra-Task Learning Transfer (ITLT)

1 Get explored states $\mathcal{M}_\varepsilon$ from $\mathcal{S}_{range}$:
2 $\mathcal{M}_\varepsilon = \{\mathcal{S}_{range} \mid \varepsilon(s) < \mathcal{C}_\varepsilon\}$
3 **if** *Exploration requirement* $\mathcal{M}_{cond}(s)$ *is met?* **then**
4 **foreach** $s_\varepsilon \in \mathcal{M}_\varepsilon$ **do**
5 Get relative distance of s_ε: $R_{dst} = |s - s_\varepsilon|$
6 **if** $s_\varepsilon > s$ **then**
7 Set weight: $W_\varepsilon(s_\varepsilon) \leftarrow e^{-R_{dst}/\theta_h}$
8 **else**
9 Set weight: $W_\varepsilon(s_\varepsilon) \leftarrow e^{-R_{dst}/\theta_l}$
10 **end**
11 **foreach** $s_\varepsilon \in \mathcal{M}_\varepsilon$ **do**
12 Normalize the weight of explored state:
$$\hat{W}_\varepsilon(s_\varepsilon) \leftarrow \frac{W_\varepsilon(s_\varepsilon)}{\sum_n^{length(\mathcal{M}_\varepsilon)} W_\varepsilon(s_n)}$$
13 **foreach** $a \in s_\varepsilon$ **do**
14 Add weighted Q-value of explored action:
15 $Q(s,a) \mathrel{+}= \hat{W}_\varepsilon(s_\varepsilon) \times Q(s_\varepsilon, a)$
16 **end**
17 **end**

$\mathcal{S}_{dst}$ distance in order to provide a balanced estimation of Q-values. In line 5, a relative distance R_{dst} between the current state (s) and explored states (s_ε) is then found. Both S_{min} and R_{dst} are positive integers below or equal to $\mathcal{S}_{dst}$. A weight $W_\varepsilon(s)$ is assigned to each explored state, which is exponentially weighted using R_{dst} (lines 7, 9).

ITLT Q-value transfer allocates higher weights to explored states with the closest proximity as the probability of states being behaviorally similar decays as R_{dst} increases. As optimal actions, i.e., DVFS points vary continuously with a state value, ITLT exploits these relationships to transfer Q-values.

Constants θ_l and θ_h control QoS-energy tradeoffs. Lower states are scaled by θ_l, while higher states are scaled by θ_h. When $\theta_h > \theta_l$, TL is more conservative prioritizing higher optimal actions and lowering the probability of violating QoS. If $\theta_h < \theta_l$, TL increases the probability of operating closer to QoS constraint (T_{perf}), but at a lower energy footprint.

Next, the obtained weight (from line 7 or 9) is normalized by the sum of weights of all explored states (line 12). Then each Q-value of the explored state (s_ε) is scaled by $\hat{W}_\varepsilon(s_\varepsilon)$ and assigned to a new corresponding Q-value of an unexplored state. This is repeated until all weighted Q-values are collected from explored states in order to obtain final Q-values.

Algorithm 2: Intra-State Learning Transfer (ISLT)

1 **if** *exploration points* ($\mathcal{A}_{ep}$) *are not defined?* **then**
2 Sort control actions $\mathcal{A}$ from min to max
3 **if** *at least* $\mathcal{A}_{min}$ *actions are available?* **then**
4 Assign initial points: $\mathcal{A}_{ep} \leftarrow [a_{min}, a_{max}]$
5 $id_{max} \leftarrow index.max(\mathcal{A})$
6 $id_{high} \leftarrow \mathcal{A}_{skip} + 1$
7 **while** $id_{high} < id_{max}$ **do**
8 **if** $\mathcal{A}[id_{high}] \neq a_{max}$*?* **then**
9 Add action: $\mathcal{A}_{ep}.append(\mathcal{A}[id_{high}])$
10 $id_{high} \leftarrow id_{high} + (\mathcal{A}_{skip} + 1)$
11 **end**
12 **else**
13 Assign min/max actions: $\mathcal{A}_{ep} \leftarrow [a_{min}, a_{max}]$
14 **if** *all actions in* $\mathcal{A}_{ep}(s)$ *explored* N_{times}*?* **then**
15 Given $perf(s,a)$ points, where $a \in \mathcal{A}_{ep}(s)$:
16 estimate linear model $f_{perf}(s,a)$
17 **foreach** $a \in \mathcal{A}(s)$ **do**
18 Find slack time: $T_{slack} \leftarrow T_{perf}$ - $f_{perf}(s,a)$
19 Estimate initial $Q(s,a)$ values using Eq. 7
20 **end**
21 **else if** *action* ($a \in \mathcal{A}_{ep}$) *is not explored* N_{times} **then**
22 Set the next action from $\mathcal{A}_{ep}$ for the exploration

5) Intra-State Learning Transfer (ISLT): ISLT is shown in Algorithm 2. When ISLT is presented to an unexplored environment with undefined exploration points ($\mathcal{A}_{ep}$), the first step is to identify actions for directed ISLT exploration. The algorithm starts by sorting control actions from the lowest to the highest (line 2), i.e., CPU frequencies in our work. There are two types of action assignments for ISLT exploration. Provided that at least $\mathcal{A}_{min}$ actions are available, the first type assigns a_{min} and a_{max} actions by default. Then, it iterates through action space ($\mathcal{A}$) appending actions to $\mathcal{A}_{ep}$ every $\mathcal{A}_{skip}$ (line 7) and dividing ISLT action space into smaller regions. Larger $\mathcal{A}_{skip}$ values, e.g., $\mathcal{A}_{skip} > 2$, exclude more actions, while lower values include more actions for exploration. For example, $\mathcal{A}_{skip} = 1$, $\mathcal{A}_{min} = 5$ and $\mathcal{A}$ has at least $\mathcal{A}_{min}$ actions, exploration points with indexes of $0, 2$ and 4 would be included in $\mathcal{A}_{ep}$. The second type assigns only a_{min} and a_{max} actions for ISLT exploration (line 13).

After actions are identified, ISLT is guided through exploration points at least N_{times} (see line 14, 21 and Fig. 2). Larger N_{times} values, (e.g., $N_{times} > 2$), improve the confidence level of estimated Q-values, but also increase the exploration requirements. For each exploration point, a temporary record of performance $perf(s,a)$ is stored for the derivation of the state's performance-action model (f_{perf}), which is required to estimate initial Q-values. ISLT reduces the complexity of models (e.g., with higher degree polynomials) for applications with non-linear performance-action behavior. It estimates f_{perf} as a linear model using the closest exploration points. For instance, given 7 exploration points (actions), where actions indexed 0, 3, 6 are explored, the performance of action 2 using f_{perf} is estimated when a linear model of f_{perf} is derived using actions with indexes of 0 and 3. This allows maintaining an acceptable level of accuracy at low complexity. Then, for each unexplored action within a state, the initial Q-value is calculated based on reward function (from Eq. (7)) and an estimated T_{slack} using a linear f_{perf} model (lines 18-19).

IV. Experimental Results

A comparative evaluation of ISIT-TQL and existing methods discussed in Section III-B is carried out firstly in terms of energy and cumulative rewards [8] of the learning agent. Additional metrics are also defined to better understand achieved QoS and relationships between the performance of the learning agent and energy-efficiency. These are defined as follows:

- ***QoS Success Rate (QSR)*** – a ratio of control cycles with satisfied QoS (i.e., $t_{perf} > T_{perf}$) and total cycles, which quantifies the achieved level of QoS.
- ***Power Normalized QSR (PNQ)*** – a ratio of QSR and average power, i.e., $QSR/Watt$. Maximum PNQ indicates the highest QoS achieved per Watt. It allows to identify the best QoS-energy tradeoffs in the observed experiments.
- ***Power Normalized Reward (PNR)*** – a metric quantified as $Reward/Watt$. PNR relates energy-efficiency and received rewards of the Q-learning agent.

1) Workloads for comparative analysis: two workloads are applied on cluster nodes for comparative evaluation. The first, called *TensorFlow-serving* (later referred as *TensorFlow*) [20], is the inference server of machine learning models used in production environments. The results are provided when serving a popular *MobileNet* [21] model for image classification. The second, is a blog website based on *WordPress* [22] framework.

Scenarios with gradual and sudden variations in service demands are applied to investigate a dynamic range of web traffic activity, i.e., user requests. In the experiments, the QoS constraint (T_{perf}) is set to 80 ms and 300 ms for *WordPress* and *TensorFlow* workloads, respectively. The selected values are aligned with hardware capabilities and QoS requirements allowing to exercise workloads with available control actions.

2) Experimental setup and parameters: an offline analysis is carried out for tuning parameters used in the experiments to provide satisfactory and robust behavior of the Q-learning agent as follows:

- common parameters include: learning rate $\alpha = 0.3$, the discount factor $\gamma = 0.4$. Reward constants $\beta_1 = 1$ and $\beta_0 = 10$. Among ISIT-TQ and *AES*, the temperature parameter (τ) and the inverse sensitivity constant (σ) are set to 1.5 and 10, respectively. This provides a balanced ratio of exploration/exploitation when performing action selection based on TD_{error} (see Eq. (5)). Also, δ parameter is inversely proportional ($1/\mathcal{A}$) to the number of actions.
- ITLT-related parameters: minimum explored states $S_{min} = 1$, state distance $\mathcal{S}_{dst} = 2$. A state is considered to be explored when $\mathcal{C}_{\varepsilon} < 0.4$. Scaling factors of lower and higher states are set to the same value ($\theta_l = \theta_h = 1$).
- ISLT-related parameters include: the number of explorations per action $N_{times} = 2$ and skipped actions $\mathcal{A}_{skip} = 1$.
- The exploration decay rate (θ) used in *CE* is set to 3.

3) Learning agent performance: Fig. 4 displays the performance of the Q-learning agent, shown as cumulative rewards, when running: a) *WordPress*, and b) *TensorFlow* workloads. The results demonstrate that our ISIT-TQL method receives the highest cumulative reward at observed control intervals,

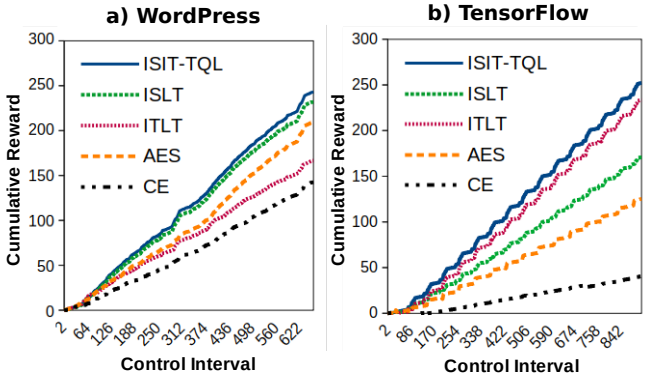

Fig. 4: Cumulative rewards under two different workloads.

and selected control actions lead to operation closer to the level set by QoS constraint (T_{perf}). This indicates that the system operates neither too fast (wasting energy) nor too slow providing the required QoS. In addition, higher cumulative rewards demonstrate better learning speed as the method is capable to identify optimal control actions at the early exploration stage. ISIT-TQL adapts to the dynamics of workloads by balancing the learning using ISLT and ITLT. For example, ISLT receives the highest cumulative rewards for the *WordPress* workload since ISLT is predominantly initiated due to more continuous explorations of state space. In contrast, *TensorFlow* workload involves more sparse exploration of the state space. This provides opportunities for exploiting the knowledge of similar states and applying ITLT. *CE* receives the least amount of rewards as it lacks adaptive controls of exploration probability, ε(s), and utilizes a randomized action selection.

4) Energy reduction: Table I includes results of evaluated methods showing: a) raw values of metrics, and b) percentage difference of each method against ISIT-TQL. The method called *default*, corresponding to Linux *Ondemand* DVFS controls, is additionally included in Table I. When compared to *default*, ISIT-TQL reduces energy by 28.67% for *TensorFlow* and by 19.06% for *WordPress* workloads. ISIT-TQL improves over *CE* by 7.3% and *AES* by 4.51% considering *WordPress* workload. *TensorFlow* workload is more compute-intensive and has higher energy requirements than *WordPress*. This makes the effect of sub-optimal control actions more pronounced in energy figures, where ISIT-TQL outperforms *CE* and *AES* by 18.06% and 14.16%, respectively.

5) QSR, PNQ and PNR metrics: with ISIT-TQL, the QSR metric, indicating QoS, is enhanced by up to 14.30% for *TensorFlow* and up to 10.43% for *WordPress* when compared to other exploration strategies. PNQ and PNR metrics given in Table I (a) are shown as non percentage values. ISIT-TQL significantly improves QoS achieved per unit of power, i.e., PNQ, by 38.41% over *default* and up to 24.93% over *CE*. The power normalized rewards of the learning agent, i.e, PNR, is drastically improved over *CE* by up to 85.94%.

6) Control interval and overheads: in the investigated applications, the performance obtained from the load balancer is a moving average of the last 1024 user requests. The delayed feedback environment allows defining a control interval in a

TABLE I: Comparative ISIT-TQL performance using different metrics under two different workload scenarios.

	a) Metrics								b) Percentage Change (%)							
Workload	**TensorFlow**				**WordPress**				**TensorFlow**				**WordPress**			
Method	**Energy (J x 10^3)**	**QSR (%)**	**PNQ**	**PNR**	**Energy (J x 10^3)**	**QSR (%)**	**PNQ**	**PNR**	**Energy**	**QSR**	**PNQ**	**PNR**	**Energy**	**QSR**	**PNQ**	**PNR**
ISIT-TQL	**51.289**	**98.245**	**0.115**	**0.097**	**34.906**	**97.378**	**0.127**	**0.482**	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
ISLT	55.585	96.153	0.104	0.064	35.132	95.737	0.124	0.457	-8.38	2.14	9.70	33.30	-0.65	1.69	2.32	5.22
ITLT	55.169	**98.245**	0.107	0.083	35.294	96.944	0.125	0.411	-7.56	0.00	7.04	13.90	-1.11	0.45	1.54	14.84
CE	60.554	84.210	0.086	0.013	37.454	87.218	0.106	0.264	**-18.06**	**14.30**	**24.93**	**85.94**	**-7.30**	**10.43**	**16.53**	**45.36**
AES	58.551	91.228	0.090	0.015	36.479	92.629	0.115	0.316	-14.16	7.24	21.44	83.86	-4.51	4.88	8.98	34.43
Default	65.993	77.860	0.070	N/A	41.559	82.630	0.090	N/A	**-28.67**	**20.76**	**38.41**	N/A	**-19.06**	**15.15**	**28.73**	N/A

range of tens of seconds. We set the control interval to 20s as it provides sufficient adaptability to be used with a wide range of workloads. A typical control cycle takes no more than 25ms to complete when running ISIT-TQL on *Intel's* i5-8250U CPU, which is a negligible control overhead.

7) Scalability and Portability: our runtime is not platform-specific, only requiring industry-standard monitors and control actions, e.g., CPU frequencies to be specified accordingly to a given system. This allows ISIT-TQL to be scaled onto existing server architectures with minimal adjustments in the configuration of the learning agent.

The instances of ISIT-TQL agent scale together with a number of managed applications deployed on virtual private servers (VPS), containers, or bare-metal servers. The key requirement is that a given application has dedicated resources to ensure robust learning enabling it to avoid re-explorations due to performance variations caused by resource sharing. ISIT-TQL dynamically scales the state space and does not require to specify its dimensions in advance.

8) Application-suitability of the method: ISIT-TQL is suitable for applications deployed in multi-process/thread server architecture environments exercised through a typical request-response communication model [19]. As the observable performance, i.e., response time is the key to learning state-action relationships rather than the servable content of applications itself, our method is generally applicable to various applications. For example, multiple instances of ISIT-TQL agent can be assigned to manage workloads of different machine learning models served by the *TensorFlow* server [20].

V. Conclusions

This work concentrates on energy minimization for distributed many-core systems under QoS constraints. The presented method, called ISIT-TQL, leverages Q-learning accelerated by proposed multi-dimensional knowledge transfer to achieve the learning of DVFS controls at higher levels of QoS. It enhances exploration by extracting knowledge of behaviorally similar states and transferring them to unexplored states known as ITLT. When "behaviorally similar" states are not available with reference to a current state, a reduced exploration at a local state's level is applied known as ISLT. The proposed method achieves energy saving of 28.67% (compared to *default* system) and 18.06% (compared to *counter* ε-*greedy*). The increased optimality of controls is demonstrated through improved QoS by up to 14.3%-20.76%. ISIT-TQL is self-sufficient and can learn the dynamics of a task's workload for both dynamic and static QoS requirements.

References

[1] C. Gu, Z. Li, H. Huang, and X. Jia, "Energy efficient scheduling of servers with multi-sleep modes for cloud data center," *IEEE Transactions on Cloud Computing*, vol. 8, no. 3, pp. 833–846, 2020.

[2] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *Journal of Grid Computing*, vol. 12, pp. 559–592, Dec 2014.

[3] S. Horovitz and Y. Arian, "Efficient cloud auto-scaling with sla objective using q-learning," in *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, pp. 85–92, 2018.

[4] B. Simmons *et al.*, "Managing a saas application in the cloud using paas policy sets and a strategy-tree," in *2011 7th International Conference on Network and Service Management*, pp. 1–5, 2011.

[5] A. Monteiro *et al.*, "Qmapper: Dynamic power and performance management in virtualized web servers clusters," in *Eighth Latin-American Symposium on Dependable Computing (LADC)*, pp. 37–46, 2018.

[6] L. S. Sousa *et al.*, "Green data centers: Using hierarchies for scalable energy efficiency in large web clusters," *Inf. Process. Lett.*, vol. 113, p. 507–515, July 2013.

[7] V. Petrucci *et al.*, "Optimized management of power and performance for virtualized heterogeneous server clusters," in *11th IEEE/ACM Int. Symposium on Cluster, Cloud and Grid Computing*, pp. 23–32, 2011.

[8] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, second ed., 2018.

[9] M. Tokic and G. Palm, "Value-difference based exploration: Adaptive control between epsilon-greedy and softmax," in *KI 2011: Advances in Artificial Intelligence* (J. Bach and S. Edelkamp, eds.), (Berlin, Heidelberg), pp. 335–346, Springer Berlin Heidelberg, 2011.

[10] F. Fernández and M. Veloso, "Probabilistic policy reuse in a reinforcement learning agent," vol. 2006, pp. 720–727, 01 2006.

[11] A. D. Tijsma *et al.*, "Comparing exploration strategies for q-learning in random stochastic mazes," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, 2016.

[12] T. Brys *et al.*, "Policy transfer using reward shaping," in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '15, (Richland, SC), p. 181–188, International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[13] Z. Zhu, K. Lin, and J. Zhou, "Transfer learning in deep reinforcement learning: A survey," 2021.

[14] R. A. Shafik *et al.*, "Learning transfer-based adaptive energy minimization in embedded systems," *IEEE Trans on CAD of Integrated Circuits and Systems*, vol. 35, no. 6, pp. 877–890, 2016.

[15] J. L. Carroll and T. Peterson, "Fixed vs. dynamic sub-transfer in reinforcement learning.," in *ICMLA*, pp. 3–8, 2002.

[16] Hardkernel, "Odroid-XU4." [Online]. Available: http://www.hardkernel.com. [Accessed: 10-Sep-2021].

[17] "HAProxy Load Balancer." [Online]. Available: http://www.haproxy.org. [Accessed: 10-Sep-2021].

[18] "Wrk 2 Load Test Tool." [Online]. Available: https://github.com/giltene/wrk2. [Accessed: 10-Sep-2021].

[19] B. Erb, "Concurrent programming for scalable web architectures," diploma thesis, Institute of Distributed Systems, Ulm University, April 2012.

[20] C. Olston *et al.*, "Tensorflow-serving: Flexible, high-performance ML serving," *CoRR*, vol. abs/1712.06139, 2017.

[21] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017.

[22] WordPress.org, "Blog tool, publishing platform and CMS." [Online]. Available: https://en-gb.wordpress.org/. [Accessed: 10-Sep-2021].