Co-optimizing Latency and Energy for IoT services using HMP servers in Fog Clusters

Sambit Shukla
sshukla@ucdavis.edu
UC Davis
Davis, CA, USA

Dipak Ghosal
dghosal@ucdavis.edu
UC Davis
Davis, CA, USA

Kesheng Wu
kwu@lbl.gov
Berkeley Lab
Berkeley, CA, USA

Alex Sim
asim@lbl.gov
Berkeley Lab
Berkeley, CA, USA

Matthew Farrens
mkfarrens@ucdavis.edu
UC Davis
Davis, CA, USA

***Abstract*—Fog computing has the potential to be an energy-efficient alternative to cloud computing for guaranteeing latency requirements of Latency-critical (LC) IoT services. However, even in fog computing low energy-efficiency of homogeneous multi-core server processors can be a major contributor to energy wastage. Recent studies have shown that Heterogeneous Multi-core Processors (HMPs) can improve energy efficiency of servers by adapting to dynamic load changes of LC-services. However, proposed approaches optimize energy only at a single server level. In our work, we demonstrate that optimization at the cluster-level across many HMP-servers can offer much greater energy savings through optimal work distribution across the HMP-servers while still guaranteeing the Service Level Objectives (SLO) of LC-services. In this paper, we present Greeniac, a cluster-level task manager that employs Reinforcement Learning to identify optimal configurations at the server- and cluster-levels for different workloads. We develop a server-level service scheduler and a cluster-level load balancing module to assign services and distribute tasks across HMP servers based on the learned configurations. In addition to meeting the required SLO targets, Greeniac achieves up to 28% energy saving compared to best-case cluster scheduling techniques with local HMP-aware scheduling on a 4-server fog cluster, with potentially larger savings in a larger cluster.**

I. Introduction

Onset of the IoT-era has triggered the deployment of sensors and the use of interactive devices in numerous domains such as vehicular networks, enterprise networks, home entertainment, and health care facilities. These domains are increasingly employing latency-critical (LC) applications such as sensor-based actuation, augmented/virtual reality, image recognition, and online translation. For these applications end-users expect and require low (sub-second) and predictable response time. Due to their growing resource-demands, LC-applications have been recently migrated from resource-constrained end-devices to remote cloud platforms [1]. But unpredictable and long network delays pose significant challenge for cloud services. The cloud providers have to guarantee the servicing of LC-requests within a much smaller latency headroom. This mandates over-provisioning of server resources, thereby sacrificing server energy efficiency [2]. Recent studies [3], [4] show that Fog Computing paradigm can address these LC-service requirements by leveraging fog's network vicinity to end-devices. Low propagation delays leave larger headroom for servicing requests.

Unlike the cloud domain, pricing models for fog computing services are not well-defined. In fact, end-users themselves might be burdened with the entire operating expenses (OpEx) for many fog-deployment use cases. Recent studies have revealed that energy usage by server CPUs is the most significant contributor [5] to the large OpEx costs in clouds. Thus several studies on cloud LC-services have proposed various CPU resource management and scheduling strategies [6]–[8] to minimize CPU energy usage while still satisfying service time objectives. Similar Latency-Energy co-optimization strategies can also be applied to improve efficiency of fog computing and reduce OpEx costs. However, unlike clouds, which mostly employ powerful homogeneous multicore-based servers, fog computing platforms enjoy the additional flexibility of using relatively energy-efficient, cheaper, but less powerful Heterogeneous multi-core processor (HMP) platforms. However, HMPs introduce additional opportunities as well as challenges for the co-optimization problem.

Recent studies [9]–[11] have found HMPs to be especially suitable for efficient execution of LC-Services. HMPs can process tasks during heavy loads with the fast but power-hungry big cores, and switch these big cores to low-power sleep states whenever slower but energy-efficient small cores can meet latency objectives for the request load. Prior works have translated the co-optimization problem into the following scheduling problem: For a given request load at a single HMP-server 1) how many LC-Service instances need to be run? and 2) on which cores (big and/or small) they need to be run? However, these proposed approaches only perform task scheduling on a small set of big and small cores within a single server. The server-level local optimization is a simple but sub-optimal approach compared to a global cluster-level approach. The energy saving could be greatly improved by viewing the entire cluster as a single server and scheduling over a large set of heterogeneous cores spread across multiple HMP servers. Our study show that using a cluster-level scheduling strategy coupled with an HMP-aware load balancing across servers saves significantly more energy while meeting the LC-request latency targets.

In this work, we present *Greeniac*, a task manager for LC-Services on HMP-based fog servers. Greeniac automatically learns the best run-time distribution of incoming LC-Service requests among HMP servers and individual cores to maximize energy savings while meeting Service Level Objectives (SLO). The contributions of our work are as follows:

1) We propose a two-level Reinforcement Learning (RL) agent to learn the optimal set of cores (from set of all cores available in the cluster) on which LC-Service instances must be hosted, at any given request load.
2) At each HMP server, we implement a Service Scheduler that employs frequency scaling, service instance scaling, and instance-to-core mapping suggested by the RL agent.
3) We design an HMP-aware Load Balancer, that exploits feedback from RL agents to perform an HMP-aware run-time request distribution across HMP servers hosting active LC-Service instances.
4) We study the impact of various system and input parameters such as cluster size, HMP configuration, LC-workload characteristic to demonstrate the advantages of Greeniac over prior works.

Our experiments show that Greeniac saves up to 28% of core energy over traditional server-level scheduling approaches on a 4-server cluster with potentially higher saving for larger clusters.

II. LC Services on Fog Clusters

Service providers have to guarantee certain SLO for LC-Services. For instance, a SLO requiring 95th percentile tail latency (or P95) target of 1 sec means service provider must ensure at least 95% of client requests are serviced within 1 sec of request arrival. To meet SLO for unpredictable request loads, service providers over-provision cluster resources resulting in high energy usage. But low energy-proportionality in server processors result in significant energy wastage by underutilized active servers [2] during non-peak traffic periods. Thus minimizing cluster energy usage under latency constraints is a challenging problem. Higher energy efficiency (requests processed per unit energy) of cluster translates to a desirable lower operational expense for service providers.

Fog computing paradigm offers several advantages and challenges for LC-Services. *Firstly*, the reduced network delays leaves larger response time headroom and offers opportunity for more energy-efficient task scheduling at fog servers. This also widens the spectrum of LC-Services to those requiring shorter response latency or higher computation overhead. Hence *latency objectives* for services need to be *re-calibrated*. *Secondly*, fog clusters typically service IoT clients in a specific domain. This reduces resource interference effects observed in cloud [6] both at the network and the server levels. However, instead of co-execution strategies [6] used to improve cloud energy-efficiency, *solo-execution strategies* have to be relied upon for fog servers. *Thirdly*, since fog deployments are on smaller scale, they can be domain-specific and use efficient, application-specific compute platforms. But the small scale and fewer number of aggregate requests hamper the adoption of auto-scaling [12] techniques used for large clouds. This also limits the scope for any service-specific customization [13]. Hence novel *application-agnostic resource management* techniques are desired for efficient service execution on fog clusters. *Finally*, due to the small client base serviced by fogs, request traffic has relatively higher variability compared to cloud. Hence the scheduling approach must *dynamically adapt* to meet the resource requirements of the *varying request load*.

III. The Case for HMPs

HMP SoCs (e.g., Snapdragon, Exynos) have been popular with mobile devices for energy-efficient processing. Some HMP-based servers (e.g. Intel QuickIA [14], Odroid-MC1 [15]) are also gaining popularity in clusters. Unlike power-hungry homogeneous multi-cores, HMPs can offer improved energy efficiency by adapting to load changes. Big-cores, with large sophisticated pipelines running at higher frequency, consume significantly higher energy compared to the smaller cores with slower in-order processing clocked at lower frequency. This architectural heterogeneity can be exploited to dynamically adapt to the varying resource requirements of LC-services.

A. Using HMPs for Fog Computing

Wide use of HMPs in mobile computing encourages its adoption in non-edge fog domain too. For fog deployments to be economically sustainable, clusters must have low infrastructural, capital and operational costs. Hence the small form factor, affordable equipment cost and low energy footprint of the modern HMP-based compute platforms are motivating factors for employing HMPs, both on mobile and fixed server platforms. Our work further demonstrates the energy-saving opportunity with HMP-cores on fog servers.

B. Using HMPs for LC-Services

Recently, use of single-ISA HMPs [14], [16] have been shown to achieve significant energy savings by employing powerful big cores during high load and low-power small cores during low loads [9]–[11]. Figure 1 depicts different scenario for load-dependent scheduling on big and small cores. A resource manager can employ a mix of two techniques to maximize energy saving: *core scaling* and *frequency scaling*.

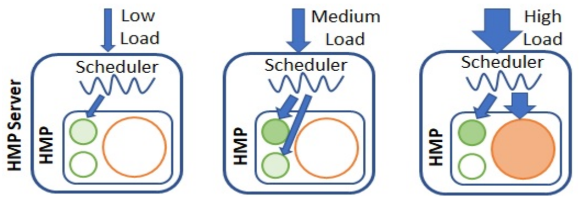

Fig. 1. Different Service Configurations (SCs) optimal for different loads. Big core enter sleep mode at low load. Core scaling with preference for small cores helps to meet SLO target optimally.

Core scaling involves scaling the number of service instances based on request load. When load increases threatening SLO target violation, additional instance may be scheduled on an idle small or big core. When the load decreases, some instances may be released and cores switched to low-power sleep modes. The option that maximizes energy saving while still meeting the SLO target for loads would be the optimal choice for a good resource manager. However identifying the optimal choice requires exploration since it would be both workload type and HMP architecture dependent.

When executing an LC-Service instance, dynamically scaling core frequency to match the request load has been shown to achieve energy savings [17], [18]. Since frequency scaling only affects dynamic power consumption of cores, savings are relatively lower compared to core scaling techniques, especially on big cores that consume high static power. Since the frequency switching time is of the order of sub-microsecond [19], this technique can be employed for LC-Services (as in Rubik [17]).

C. Cluster-level energy saving opportunity

While server-level resource management can achieve energy saving on a HMP server, in a cluster with multiple servers, a server-local task/resource manager can only achieve a local energy optimum. We believe, energy savings could be significantly increased with a cluster-level resource manager that is aware of all the compute-resources available in the cluster. To maximize energy savings, two techniques must be tuned in tandem: service scheduling and load balancing.

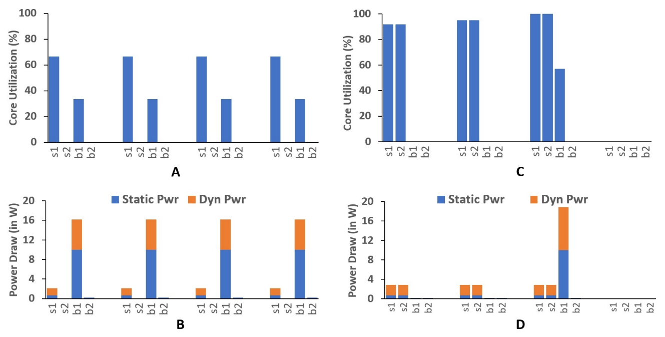

Fig. 2. Energy saving opportunity on a 2B-2S HMP-server cluster @50rps. Server-level optimization (left) with 12.5rps per server requires 4B and 4S cores per cluster with low core utilization. Cluster-level optimization (right) uses fewer big cores (1B and 6S cores) with high core utilization and lower aggregate energy.

1) Service scheduling: At the cluster-level, number of possible combinations big and small cores are large. A cluster task scheduler that could identify the most efficient core combination for the current cluster load and elastically initiates or terminates the services on those cores/servers can maximize energy efficiency of the cluster. To verify, we subjected a load of 50 requests per second (rps) for Apache Lucene to a 4-server cluster with 2B-2S HMP configurations. Assuming a HMP-aware local task-scheduling (as in [9]), requests were equally distributed as per least-loaded strategy [20]. Fig 2(A) shows 2 service instances (1 big and 1 small core) required to meet SLO at each server. Additionally core utilization is low and there is significant energy wastage (Fig 2(B)). With cluster-level HMP-awareness, as implemented in Greeniac, a cluster-level efficient configuration consisting of more small cores and fewer big cores (Fig. 2(C)) can be used to host services, meet SLO and lower aggregate cluster energy usage (as shown in Fig. 2(D)).

2) Load balancing: Once efficient core combination is determined, it is important to optimally divide the aggregate cluster load across the instances to guarantee the latency requirements. Load balancing of requests across HMP servers ought to be proportional to each servers aggregate service capacity. Additionally, if each server meets the tail latency SLO for its hosted service instances, aggregate latency objective for the entire cluster is implicitly met. In our work, we dynamically identify the optimal tuning options for both techniques within a cluster-wide scheduler to maximize cluster energy savings.

IV. Greeniac: A smart task manager

Greeniac consists of three primary components: a *learning agent*, a *service scheduler*, and a *load balancer*. The *learning agent* employs a two-level Reinforcement Learning (RL) approach and comprises of a Server-level Learning Agent (SL-Agent) that runs on the individual HMP-servers and a Cluster-level Learning Agent (CL-Agent) which runs on the Orchestrator Node. The overall architecture is shown in Figure 3. For different request loads, the SL-Agent learns the service configuration (SC) that minimizes the energy usage while meeting tail latency requirement for LC-Services. A SC is a list of active cores in the server and their corresponding frequencies. This information is then used by the CL-Agent to learn the best cluster-wide service configuration (CSC) for different aggregate request loads observed by the cluster. CSC is the collection of optimal SCs for all servers in the cluster. The *Service scheduler* dynamically employs service management at server-level to implement configuration determined by the *learning agent*. The *load balancer* is responsible for distributing the requests among the active servers. The three components execute in a coordinated fashion to maximize energy savings of cluster under a given SLO target. We describe some details of these components in following subsections.

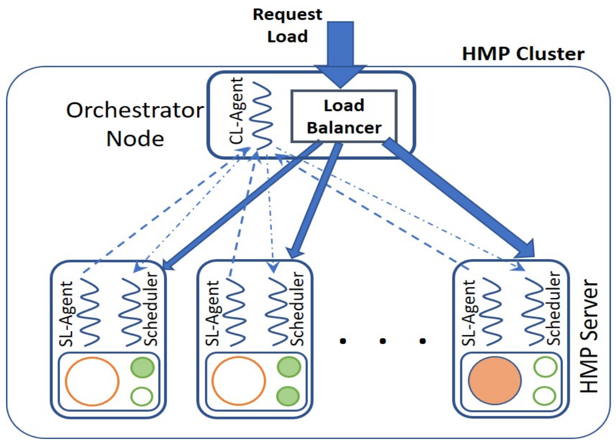

Fig. 3. Greeniac task management on a HMP-cluster. Server-level SL-Agents learn optimal local SC and update to CL-Agent, which then learns optimal CSC. CL-Agent drives local Service Schedulers and central Load Balancer to activate the optimal CSC and distributes load proportionally every epoch.

A. Learning Agent

We formulate our energy optimization problem as a *Multi-Armed Bandit (MAB) problem* [21]. MAB consists of a single state (bandit) and multiple actions (arms) wherein an agent learns for each state which action returns maximum reward.

The Learning Agent employs a 2-level learning strategy. In first stage, it employs SL-Agent at each server to learn the optimal SCs. For this stage, a state is defined to be the average arrival rate of LC-Service requests at a server. The action is the choice of a SC in the server. Note that in our work, we assume that each active core hosts a single LC-Service instance. After taking an action, the agent waits for a pre-defined time-period to measure the achieved tail latency and energy usage based on which it calculates the reward. The SL-agent learns the best action (SC) for the current state (load) from exploration of various action choices in run-time. Optimal SCs learned by SL-Agents are reported to the CL-Agent. The CL-Agent learns the optimal CSC for each load by employing similar learning strategy. For this stage, state is defined by average arrival rate at the cluster and action is a choice of CSC. The CSC with highest calculated reward, is identified as the optimal CSC for a given cluster load. Post learning completion, optimal CSCs denote the cluster-wide list of cores (and their corresponding frequencies) that need to host LC-Service instances in order minimize energy usage of entire cluster while still meeting the SLO target for current request load at the cluster.

The Learning Agent uses several heuristics (e.g. for, action-space pruning, identifying initial action for learning, choosing next action for learning based on current reward, etc.) to speed up the learning process, both in SL-Agent and CL-Agent. The state space is discretized into fine-grained bin sizes to avoid burden of continuous space learning. Reward functions in both stages take the measured P95 and energy usage into account to calculate the reward for each state-action pair. Details regarding the learning algorithm, the optimization techniques used and their implementation can be found in [22]. Though numerous other learning-based approaches exist, comparing them with our proposed learning strategy is beyond the scope of this paper.

B. Service Scheduler

The SC update received by SL-Agents at each epoch consists information about the number of big and small cores within a HMP to be activated and their corresponding core frequencies. SL-Agent relays this information to the Service Scheduler module. The scheduler is responsible for service instance creation/deletion on specified and core frequency scaling. When an updated SC requires an additional core, a new LC-Service instance is created and affinitized to the new core by SS. This closely resembles a light-weight container instantiation in cloud/fog domain. Subsequently any new or waiting request is scheduled on the new service instance in a FCFS fashion i.e. the service instance with longest idle period receives next request. For removing an existing core from prior SC, no additional requests are scheduled to the corresponding service instance and any executing request is allowed to be serviced till completion. Subsequently the instance is deleted by Scheduler when the instance becomes inactive. The Scheduler also performs frequency scaling on the active cores based on updated SC. Since frequency scaling using Linux governor takes only microseconds to activate, the update and corresponding scaling effect is almost instantaneous.

C. Load Balancer

A Load Balancer (LB) routine is responsible for distributing requests arriving at the gateway server across the HMP servers. The CL-Agent maintains the average throughput information of each explored service configuration during the learning phase. Upon identifying the optimal next epoch CSC based on last epoch load, it shares the throughput information for each HMP-server's SC with the LB. LB infers the load fraction of each server based on the throughput ratios between them. It then fits the ratios in to normalized scale window of 0 to 1. For every request, LB generates a random number between 0 and 1, identifies which server it corresponds to from the load fraction window and forwards the request to the server.

This throughput-proportional load balancing is crucial for energy-optimal load processing at individual HMP-servers. Since each server is configured for load-specific optimal SCs for the next epoch, an improper balancing can cause tail violations in some servers and lower efficiency in others. However as shown in our studies later (Fig.5), the random number generation approach balances the load across all HMP-servers optimally and meets the latency requirements at all serviceable load ranges for the cluster.

V. Experimental Approach

Since public cloud currently offer only homogeneous processor based server instances, we had to rely on simulations to study the effectiveness of Greeniac. Furthermore, none of the available cloudlet simulators (e.g., CloudSim [23] or its extensions) support HMP servers. To support HMP-aware execution and dynamic energy usage estimation for the cluster, we developed a custom in-house simulator, HMP-ClusterSim (described below), based on SimPy and instrumented it with our real system measurements.

For our experiments, we also addressed the fog-specific challenges discussed in Section II. We performed our experiment on small clusters representative of fog clusters. We avoided co-execution scenarios by dedicating a single core for each LC-Service instance. Latency objectives were also adjusted to a longer SLO of 1 sec (instead of sub-second).

A. HMP-ClusterSim Simulator

Our simulator mimics a typical high-bandwidth cluster with negligible network delay. Like CloudSim, our simulator is event-driven. A single server instance acts as a gateway (or broker [23]) and remaining servers as single-HMP nodes hosting a LC-Service instance (or Task cloudlets [23]) at each core (1 core per VM/container). We assume sufficient memory and network bandwidth at each server and neglect intra- and inter-instance interference. The gateway server generates requests following a Poisson process at specified mean rate; requests are distributed across the nodes following our load balancing logic (Section IV-C). Within each server, a service scheduler distributes requests from a centralized queue using FIFO among the scheduler-activated cores (Section IV-B). To model service time variations, task length follows a log-normal distribution (μ=0, σ=0.25) scaled by the average service time. The number of big and small cores within a HMP-server and their individual throughput are assumed to scale linearly with frequency scaling, ignoring memory hierarchy bottlenecks and interference. Responses are sent back to gateway server to record the response latency. P95 for all requests received per sampling epoch (set at 100s) is calculated at the end of every epoch. P95 of 1 sec is set as the default SLO. For RL, each state represents a bin of 10 requests per second (rps). The results discussed below are based on a 4-server cluster with each server containing a single HMP with 2 big (2B) and 2 small (2S) cores.

B. *Incorporating real system measurements*

We performed our initial study on a high-end Xeon E5-2637 (big-core) powered Dell Power-Edge Server and an Atom C3558 (small-core) powered Super-micro Server. From both platforms, we collected the averaged service time, throughput, idle power and dynamic power values at various core frequency levels for a single service instance of Apache Lucene. These data were used in the model of the big and small core in HMP-ClusterSim. These statistics will vary with the type of processors, server configuration and application. Though, we use the representative values for a single setup to demonstrate the effectiveness of our proposed approach, scaling and sensitivity studies performed for variations in server configuration and application characteristics are presented in the following section.

.

VI. Results and Observations

In this section we first demonstrate the higher energy saving achieved by Greeniac compared to popular cluster scheduling approaches. Then we study the impact of scaling and LC-Service application characteristics on energy saving.

A. *Cluster-level saving*

To demonstrate the gains from cluster scheduling we compared Greeniac against two popular cluster scheduling approaches: bin-packing and least-loaded. Bin-packing attempts to maximize utilization of each active server before scheduling services on an inactive server. On the contrary least-loaded dispatches a request to the least-loaded server thereby trying to distribute services and request equally across all servers. For a fair comparison, we implemented the Hipster learning [9] at the individual server level for both scheduling approaches to ensure scheduling at each server to be HMP-aware, SLO guaranteed and energy-optimal for the observed server load.

Figure 4 shows the energy usage for the 3 approaches normalized by the maximum cluster energy usage for HMPs.

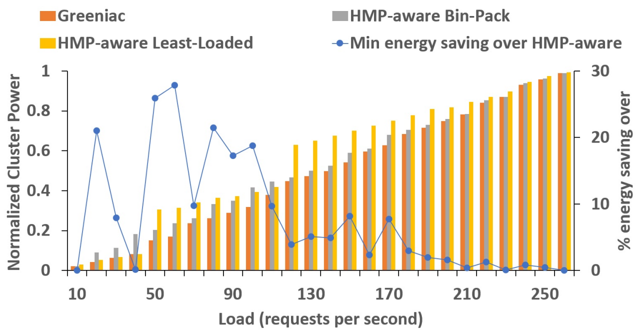

Fig. 4. Normalized energy usage for various HMP-aware scheduling approaches. Greeniac has higher energy saving (up to 28%) over locally HMP-aware approaches for all load ranges

The line shows the percentage energy saving of Greeniac with respect to the most efficient locally HMP-aware scheduling strategy at the corresponding load. Greeniac achieved a higher energy saving for a large range of load values. Greeniac gained high HMP energy saving for the low and mid range load values that typically represent the majority of load traffic periods. At very high loads, most HMP cores in all servers execute LC-Service instances with high utilization and all cluster scheduling techniques perform almost identically. At low loads, least-loaded distributes tasks equally across all 4 servers each receiving 1/4th of cluster load, which can be serviced by small cores within tail latency (TL) constraints. As load is increased, at a certain threshold cluster load level, every server requires to schedule service on big cores to meet its tail target, resulting in poor power savings due to a large number of under-utilized big cores being active across the cluster. On the contrary, bin-packing tries to maximize core utilization for active cores resulting in fewer big cores being active across the cluster. However, since bin-packing tries to maximize utilization of all cores within a HMP before scheduling tasks on an inactive HMP, it schedules services on big cores of active HMPs while ignoring availability of low power cores on inactive servers.

We observed that Greeniac achieves up to 28% higher energy saving over the best case server-level HMP-aware scheduling. Figure 5 (above) shows the active cores across all 4 servers for all tested loads. Each line pair represents the number of big and small cores activated at a given load on each of the 4 servers. We observe that Greeniac employs fewer big cores across the entire cluster and tries to maximize utilization of all active cores. Figure 5 shows the request distribution across the 4 servers with a proportionally distributed across the servers based on the service capacity. For instance, at 100rps, HMP1 has 2 big (2B) and 2 small (2S) cores active and processes around 70% of cluster load, whereas rest of the HMPs have only 2 small (2S) active and each process only 10% of the load.

B. *Scalability Analysis*

1) Processor heterogeneity: Greeniac achieve higher energy savings for HMPs with more small cores hence in Fig.

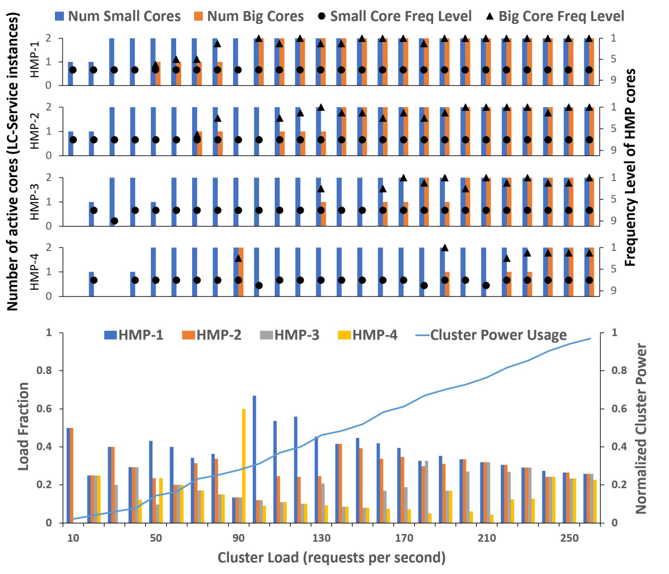

Fig. 5. Cluster dynamics corresponding to Fig 4. (a) Number of active cores and their frequency levels for CSCs learned by Greeniac (above). Each active core hosts a single LC-Service instance. (b) Load distribution proportional to the service capacity of each server SC (below).

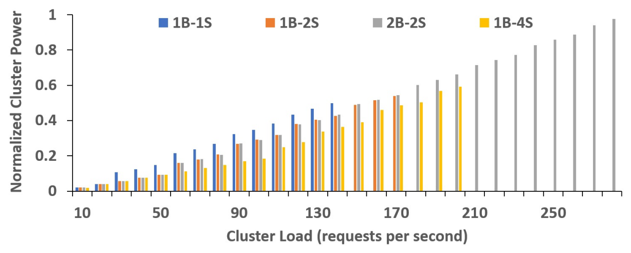

Fig. 6. Greeniac-enabled power usage for various HMP-configurations in a 4-server cluster. HMPs with more small cores enable greater cluster power saving

6, 1B-4S maximizes saving. Since Greeniac selects optimal cluster configuration from a global pool of HMP cores, HMPs with same number of small cores (e.g. 1B-2S and 2B-2S) achieve similar cluster energy savings. However, larger number of big cores with greater service capacity (as in 2B-2S HMPs) can meet tail latency requirement for high cluster loads.

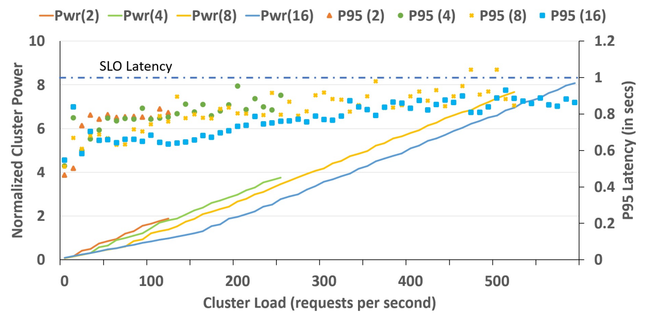

Fig. 7. Normalized Power usage and Tail latency for various cluster sizes in a 2B-2S HMP cluster. Greeniac can achieve greater energy saving with larger cluster sizes by exploiting a larger set of small cores.

2) Cluster size: We observed that larger clusters that can service higher loads also offer opportunity for more energy saving with Greeniac. In Fig. 7 at 100 rps, a 16-server cluster consumes least energy compared to a smaller sized cluster. At higher load, Greeniac learns to utilize a larger set of small cores available in larger clusters to meet the SLO targets, if possible. At low loads, different sized clusters consume similar energy since Greeniac learns to activate small cores on fewer servers. For e.g. Greeniac learns that, at 20rps, 2 small cores each on 2 servers can meet the SLO targets irrespective of cluster size, thereby resulting in same aggregate energy usage. As load increases, Greeniac schedules service on big cores of one or more active servers on smaller clusters. Thus at any given load, smaller clusters have higher number of big cores hosting LC-Services.

With increase in heterogeneity, HMP size and cluster size, the state space for the learning agent expands exponentially. However, our optimizations [22] enable fast learning on a typical fog cluster with several LC-service instance (as shown above). Once learnt, an optimal-CSC can be applied to the cluster by Greeniac almost instantly (few microseconds).

C. Impact of LC-Service Characteristics

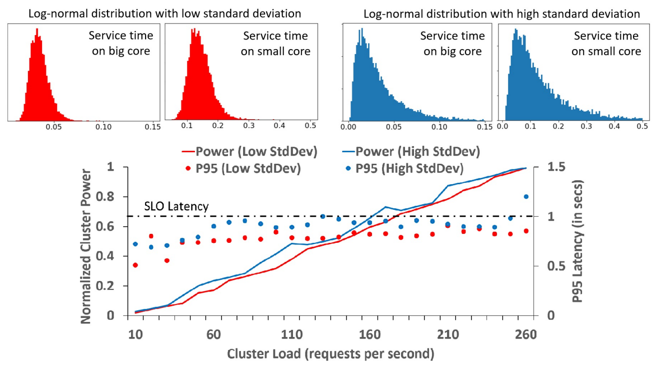

Fig. 8. Effect of service time distribution on aggregate HMP power. Greeniac learns to prefer big cores to meet target SLO for services with higher standard deviation, thus consuming higher cluster power.

1) Service time distribution: LC-Services may vary in service time distribution. Our default test service had a low standard deviation (= 0.25). In Fig 8 we compared against a service with identical mean service time but higher deviation (= 0.8). We observed that Greeniac consumes more energy for LC-Services with larger standard deviation. Larger service time variations implies a wider service time range on a small core. Therefore a request scheduled to be serviced on a small core is more likely to violate SLO target. Greeniac learns this service property and finds service configurations with big cores more suitable for higher service time deviation. This results in higher cluster energy usage compared to services with lower service time deviation. Increased variability of service latency is also the reason behind the variation in tail latency and violation of SLO at a relatively lower load as shown in Fig 8.

2) Application Characteristics: Compute and memory characteristics of services can lead to different power characteristics and average service time ratios between big and small cores (B:S). Compared to other applications, compute-bound applications are known to compute relatively much faster on big cores [24]. Whereas service time for memory bound applications are relatively less slower on small cores since applications spend a large number of CPU-independent idle cycles on memory waits. Service time ratio between big and small cores and the power characteristics can be affected by other factors such as cache characteristics, memory access pattern, compute units used, etc. However, assuming applications have identical power characteristics i.e., energy consumed for a given load, average service time ratio (B:S) is the only application-specific factor that affects the optimal configuration choice.

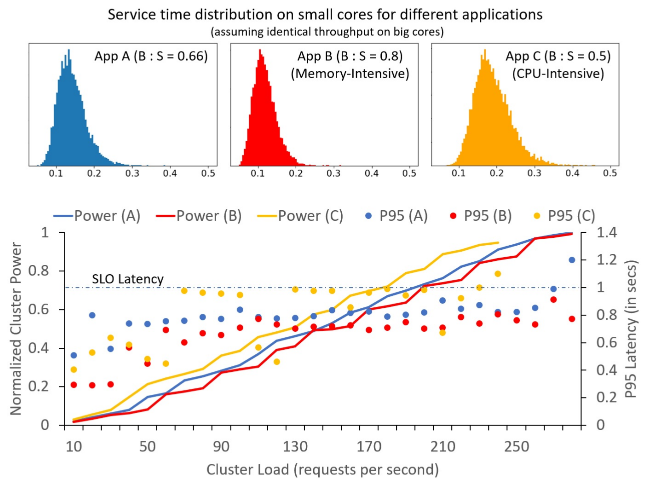

Fig. 9. Effect of application characteristics on Cluster HMP power and Tail Latency. CPU-intensive services run relatively slower while memory-intensive services run relatively faster on small cores w.r.t big cores (above). Greeniac learns to prefer big cores for CPU-intensive and small cores for memory-intensive (below).

We consider 3 applications with different B:S ratios at 1200 MHz core frequency: our default application (Lucene) with B:S as 0.66, a relatively more compute intensive service (B) with lower B:S ratio of 0.5 and a relatively more memory intensive service (C) with B:S of 0.8. Assuming identical service characteristics for the big core, B has a wider and C has a narrower service time range on small core. Owing to lower service rate for B, slow cores violate the SLO targets for a lower load compared to A and thus aggregate load serviceable by cluster is lower. On the contrary, slow cores can service higher loads for C, leading to a higher cluster load support as shown in Fig. 9. Greeniac implicitly learns these application characteristics and identifies the optimal configuration for each application service at various loads. It consumes lower cluster energy for C by exploiting small cores, and consumes higher cluster energy for B due to relatively greater use of big cores.

3) SLO latency: We made Greeniac learn optimal configurations for our test service with different SLOs. For a SLO of 500ms, the 4-server cluster could only service loads up to 100rps before violating SLO targets. Greeniac had to rely on big cores and consumed much higher energy compared to the default 1s SLO, to meet the stricter latency target. Since CPU energy usage is proportional to utilization, we observe that, for 0.5s SLO, cluster utilization could barely reach 60% before violating SLO margin. Thus stricter SLOs cause Greeniac to use big cores with low utilization and leads to higher energy usage and lower cluster efficiency. We also observed that an extremely lenient SLO of 10s does not gain any significant energy saving for the cluster. Rather that makes it harder for Greeniac to learn the optimal configuration that can meet latency targets. As shown in Fig 10, the tail latency for the optimal configuration inferred by Greeniac for the 10s SLO case is irregular for various loads. This is because a long latency target allows for a long request wait queue and can render the system unstable if service rate drops below arrival rate. Besides, for large SLOs, a slight variations in request load can greatly affect the tail latency and hence the reward.

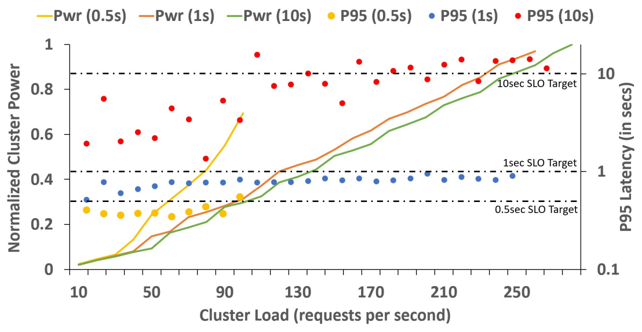

Fig. 10. Effect of SLO target on average cluster power and tail latency. Greeniac learns to use big cores at stricter tail latency targets, thus consuming higher energy and supporting lower load. Very high latency margins does not gain proportional energy saving, rather makes system unstable and thus hard to learn for optimality.

VII. Related Works

Resource scaling [7], [17] and resource sharing [6], [8], [17], [25] techniques proposed in recent years for improving energy efficiency of LC-Services have limited effectiveness due to lack of energy-proportionality in server processors. More recent works such as Hipster [9], Octopus-Man [10] and Haque et.al. [11] propose use of HMPs, which have proven to be energy-efficient and widely deployed on mobile platforms. Octopus-Man [10] uses a PID controller to estimate the HMP service configurations based on recently observed tail latency. Hipster [9] models the server system as a MDP to learn optimal HMP configurations that meet the tail target for a single server. Haque et.al. [11] employ PID controller to estimate the threshold latency for switching requests from smaller to bigger cores. However these works try to perform energy-aware scheduling only at a server-level, thereby leaving scope for significant saving at cluster-level achieved by Greeniac.

Energy-aware task scheduling has been widely studied [26] for homogeneous clusters. Auto-scaling [12] techniques with bin-packing scheduling has been popularly used in commercial cloud platforms to reduce aggregate server level energy. Such scaling has shown to gain significant energy saving for low energy-proportional (EP) servers by powering off idle servers. However with servers becoming more energy-proportional [27], energy saving from such scaling techniques are significantly lower with non-EP CPUs becoming the dominant power consumer for rack machines. HMPs can further improve the EP for such future servers. But due to lack of HMP usage in clusters, such studies have not been performed for HMP-servers. Energy efficient scheduling on HMPs have been studied in [28] but do not consider tail latency targets.

Heterogeneous architectures such as GPUs [29] and FPGAs have been employed to host LC-Services. Such accelerators gain order of magnitude improvement in latency and energy saving for applications. But they are only suitable for specific applications and require enormous porting effort making them unsuitable for generic online services.

VIII. Conclusions and Future Work

We developed a cluster level task manager, Greeniac, for fog-hosted LC-Services that uses reinforcement learning to learn to achieve a minimal energy utilization while guaranteeing tail latency constraints. We exploit the processor heterogeneity in a HMP to save energy using small cores instead of big cores at a cluster level. Our work saves significant energy over previous works performing HMP-aware local scheduling with traditional cluster schedulers. We performed several tests to demonstrate the advantages of Greeniac.

Due to the generic nature of Greeniac, it can be extended to HMP-based heterogeneous clusters and non-HMP clusters. Our scaling studies demonstrate increased energy saving for larger clusters. On larger clusters hosting even more LC-Service instances, a hierarchical load balancing approach may be employed to overcome practical scaling issues. We leave these incremental studies for our future work.

References

[1] F. Li, M. Vögler, M. Claeßens, and S. Dustdar, "Efficient and scalable iot service delivery on cloud," in *2013 IEEE sixth international conference on cloud computing*. IEEE, 2013, pp. 740–747.

[2] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," 2007.

[3] J. Dolezal, Z. Becvar, and T. Zeman, "Performance evaluation of computation offloading from mobile device to the edge of mobile network," in *2016 IEEE Conference on Standards for Communications and Networking (CSCN)*. IEEE, 2016, pp. 1–7.

[4] V. Valancius, N. Laoutaris, L. Massoulié, C. Diot, and P. Rodriguez, "Greening the internet with nano data centers," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 2009, pp. 37–48.

[5] A. Shehabi, S. Smith, D. Sartor, R. Brown, M. Herrlin, J. Koomey, E. Masanet, N. Horner, I. Azevedo, and W. Lintner, "United states data center energy usage report," 2016.

[6] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: Improving resource efficiency at scale," in *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3. ACM, 2015, pp. 450–462.

[7] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards energy proportionality for large-scale latency-critical workloads," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. IEEE, 2014, pp. 301–312.

[8] C. Delimitrou and C. Kozyrakis, "Hcloud: Resource-efficient provisioning in shared cloud systems," *Acm Sigplan Notices*, vol. 51, no. 4, pp. 473–488, 2016.

[9] R. Nishtala, P. Carpenter, V. Petrucci, and X. Martorell, "Hipster: Hybrid task manager for latency-critical cloud workloads," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 409–420.

[10] V. Petrucci, M. A. Laurenzano, J. Doherty, Y. Zhang, D. Mosse, J. Mars, and L. Tang, "Octopus-man: Qos-driven task management for heterogeneous multicores in warehouse-scale computers," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 246–258.

[11] M. E. Haque, Y. He, S. Elnikety, T. D. Nguyen, R. Bianchini, and K. S. McKinley, "Exploiting heterogeneity for tail latency and energy efficiency," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 625–638.

[12] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. A. Kozuch, "Autoscale: Dynamic, robust capacity management for multi-tier data centers," *ACM Transactions on Computer Systems (TOCS)*, vol. 30, no. 4, p. 14, 2012.

[13] C.-H. Hsu, Y. Zhang, M. A. Laurenzano, D. Meisner, T. Wenisch, J. Mars, L. Tang, and R. G. Dreslinski, "Adrenaline: Pinpointing and reining in tail queries with quick voltage boosting," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 271–282.

[14] N. Chitlur, G. Srinivasa, S. Hahn, P. K. Gupta, D. Reddy, D. Koufaty, P. Brett, A. Prabhakaran *et al.*, "Quickia: Exploring heterogeneous architectures on real prototypes," in *IEEE International Symposium on High-Performance Comp Architecture*. IEEE, 2012, pp. 1–8.

[15] "Odroid heterogeneous multi-core cluster and home cloud," https://magazine.odroid.com/article/odroid-hc1-and-odroid-mc1/.

[16] B. Jeff, "Big. little system architecture from arm: saving power through heterogeneous multiprocessing and task context migration," in *Proceedings of 49th Design Automation Conference*. ACM, 2012, pp. 143–146.

[17] H. Kasture, D. B. Bartolini, N. Beckmann, and D. Sanchez, "Rubik: Fast analytical power management for latency-critical systems," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2015, pp. 598–610.

[18] G. Prekas, M. Primorac, A. Belay, C. Kozyrakis, and E. Bugnion, "Energy proportionality and workload consolidation for latency-critical applications," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*. ACM, 2015, pp. 342–355.

[19] A. Mazouz, A. Laurent, B. Pradelle, and W. Jalby, "Evaluation of cpu frequency transition latency," *Computer Science-Research and Development*, vol. 29, no. 3-4, pp. 187–195, 2014.

[20] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: flexible, scalable schedulers for large compute clusters," in *ACM European Conference on Computer Systems*, 2013.

[21] R. S. Sutton, A. G. Barto, F. Bach *et al.*, *Reinforcement learning: An introduction*. MIT press, 1998.

[22] S. Shukla, https://github.com/sambitshukla/Greeniac.

[23] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[24] M. A. Suleman, M. Hashemi, C. Wilkerson, Y. N. Patt *et al.*, "Morphcore: An energy-efficient microarchitecture for high performance ilp and high throughput tlp," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2012, pp. 305–316.

[25] C. Delimitrou and C. Kozyrakis, "Quasar: resource-efficient and qos-aware cluster management," in *ACM SIGARCH Computer Architecture News*, vol. 42, no. 1. ACM, 2014, pp. 127–144.

[26] S. Singh and I. Chana, "A survey on resource scheduling in cloud computing: Issues and challenges," *Journal of grid computing*, vol. 14, no. 2, pp. 217–264, 2016.

[27] D. Wong, "Peak efficiency aware scheduling for highly energy proportional servers," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2016, pp. 481–492.

[28] E. D. Sozzo, G. C. Durelli, E. Trainiti, A. Miele, M. D. Santambrogio, and C. Bolchini, "Workload-aware power optimization strategy for asymmetric multiprocessors," in *Proceedings of the 2016 Conference on Design, Automation & Test*. EDA Consortium, 2016, pp. 531–534.

[29] H. Yang, Q. Chen, M. Riaz, Z. Luan, L. Tang, and J. Mars, "Powerchief: Intelligent power allocation for multi-stage applications to improve responsiveness on power constrained cmp," in *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2. ACM, 2017, pp. 133–146.