Check for updates

ACM DIGITAL LIBRARY

acm open

Latest updates: https://dl.acm.org/doi/10.1145/3712285.3759861

RESEARCH-ARTICLE

GreenMix: Energy-Efficient Serverless Computing via Randomized Sketching on Asymmetric Multi-Cores

ROHAN BASU ROY, The University of Utah, Salt Lake City, UT, United States

TIRTHAK PATEL, Rice University, Houston, TX, United States

BAOLIN LI, Netflix, Inc., Los Gatos, CA, United States

SIDDHARTH S SAMSI, Lincoln Laboratory, Lexington, MA, United States

VIJAY GADEPALLY, Lincoln Laboratory, Lexington, MA, United States

DEVESH TIWARI, Northeastern University, Boston, MA, United States

Open Access Support provided by:

Rice University

The University of Utah

Northeastern University

Lincoln Laboratory

Netflix, Inc.

PDF Download
3712285.3759861.pdf
23 February 2026
Total Citations: 1
Total Downloads: 2181

Published: 16 November 2025

Citation in BibTeX format

SC '25: The International Conference for High Performance Computing, Networking, Storage and Analysis
November 16 - 21, 2025
MO, St. Louis, USA

Conference Sponsors:
SIGHPC

https://doi.org/10.1145/3712285.3759861
ISBN: 9798400714665

GreenMix: Energy-Efficient Serverless Computing via Randomized Sketching on Asymmetric Multi-Cores

Rohan Basu Roy
University of Utah
Salt Lake City, USA
rohanbasuroy@sci.utah.edu

Tirthak Patel
Rice University
Houston, USA
tirthak.patel@rice.edu

Baolin Li
Netflix
San Francisco, USA
li.baol@northeastern.edu

Siddharth Samsi
MIT Lincoln Laboratory
Boston, USA
sid@ll.mit.edu

Vijay Gadepally
MIT Lincoln Laboratory
Boston, USA
vijayg@mit.edu

Devesh Tiwari
Northeastern University
Boston, USA
d.tiwari@northeastern.edu

Abstract

GreenMix is motivated by the renewed interest in asymmetric multi-core processors and the emergence of the serverless computing model. Asymmetric multi-cores offer better energy and performance trade-offs by placing different core types on the same die. However, existing serverless scheduling techniques do not leverage these benefits. GreenMix is the first serverless work to reduce energy and serverless keep-alive costs while meeting QoS targets by leveraging asymmetric multi-cores. GreenMix employs randomized sketching, tailored for serverless execution and keep-alive, to perform within 10% of the optimal solution in terms of energy efficiency and keep-alive cost reduction. GreenMix's effectiveness is demonstrated through evaluations on clusters of ARM big.LITTLE and Intel Alder Lake asymmetric processors. It outperforms competing state-of-the-art schedulers, offering a novel approach for energy-efficient serverless computing.

CCS Concepts

• **Computer systems organization → Cloud computing**.

Keywords

Serverless Computing, Energy Efficiency, Asymmetric Multi-Cores

ACM Reference Format:
Rohan Basu Roy, Tirthak Patel, Baolin Li, Siddharth Samsi, Vijay Gadepally, and Devesh Tiwari. 2025. GreenMix: Energy-Efficient Serverless Computing via Randomized Sketching on Asymmetric Multi-Cores. In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '25), November 16–21, 2025, St Louis, MO, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3712285.3759861

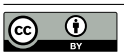
1 Introduction

Background and Motivation. This work aims to exploit the renewed interest in asymmetric multi-core processors (CMPs) for an emerging computing model: serverless computing. Previous research shows that not all cores on multi-core chips are utilized in both data centers and IoT/edge devices due to demand variations, resulting in under-utilization and energy waste [24, 38, 43, 58, 64]. This has motivated a renewed interest in asymmetric CMPs (e.g., Intel Alderlake, ARM big.LITTLE, Samsung Exynos) [50, 59, 60] which put different types of cores on the same die (instead of identical cores as in traditional symmetric/ homogeneous CMPs) – (1) *performance cores* (`PCores`) provide better performance but higher energy consumption, and (2) *efficiency cores* (`ECores`) have lower energy consumption but worse performance. These asymmetric designs are attractive because they yield higher energy efficiency – aligned with emerging carbon-footprint considerations [16, 26, 72].

Serverless computing has gained significant interest due to its simplicity, flexibility, and manageability for end users. It is a function-as-a-service computing paradigm, where users express their tasks as stateless functions, which the cloud provider executes without exposing hardware details. Serverless is gaining popularity in production HPC clusters [11] for its ability to elastically scale resources, reduce queue delays, and support modular, event-driven workflows without requiring static resource provisioning [10, 52, 53]. However, these serverless functions often incur high startup latency (referred to as *cold start*) [22, 54, 55]. To mitigate this, providers keep functions alive in memory for a certain period, in anticipation of future invocations of the same function. If the future invocation arrives within the *keep-alive time (KAT)*, the function execution starts without incurring the cold-start overhead (referred to as *warm start*). To improve the service time (which includes both execution time and potential cold start overhead), the providers employ the keep-alive policy to increase the chances of warm starts, but this incurs significant *keep-alive cost (KAC)* – often a primary serverless optimization focus [54, 55, 63, 70]. Our experimental results (Sec. 3) show that *asymmetric CMPs* (with `PCores` and `ECores`) offer significant energy savings for serverless functions, compared to a *homogeneous/symmetric CMP*. Exploiting this opportunity requires careful design that optimizes for serverless-specific characteristics.

GreenMix Key Ideas and Contributions. GreenMix is the first, novel serverless function warm up, placement, and execution technique to exploit processor asymmetry on CMPs for serverless workloads. *GreenMix's goal is to minimize energy consumption and keep-alive cost for serverless workloads while meeting QoS targets.*

While processor asymmetry provides the opportunity for energy savings, exploiting this opportunity for serverless functions is challenging due to the need for co-optimization of keep-alive



This work is licensed under a Creative Commons Attribution 4.0 International License.
SC '25, St Louis, MO, USA
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1466-5/25/11
https://doi.org/10.1145/3712285.3759861

cost, energy, and performance in a high-dimensional optimization search space. Effective warm up, placement, and keep-alive time decisions for a function need to consider the characteristics of other simultaneously-invoked serverless functions because different functions within the pool of simultaneously-invoked functions provide different relative degrees of opportunity for energy and keep-alive cost savings, and have different QoS targets. Unfortunately, the pool of simultaneously invoked functions is often large, and these functions need to be co-optimized (high-dimensional optimization space) to exploit their relative opportunities. These functions also have diverse re-invocation and execution characteristics (including PCore/ECore friendliness) – pool of simultaneously invoked functions changes over time, making the optimization challenging.

To address these challenges, GreenMix employs randomized sketching to map information from high-dimensional space to low-dimensional space [37, 49]. In this low-dimensional space, GreenMix probabilistically optimizes the serverless-specific optimization parameters (e.g., PCore/ECore friendliness for energy savings and keep-alive time for keep-alive cost savings). Then, GreenMix projects back the information to the original high-dimensional space. Leveraging this, GreenMix prepares intelligent "function mixes" of warmup-eligible functions on different nodes to match the degree of asymmetry on the nodes (i.e., number of PCores and ECores). These function mixes correspond to a set of functions with different PCore/ECore friendliness and memory consumption needs. GreenMix's optimized keep-alive time also helps functions reduce their cold-start times, which in turn improves their chances to meet the QoS targets. Finally, upon function invocation, GreenMix probabilistically assigns a core for function execution to meet its QoS target and achieve energy savings. We demonstrate that, even without employing dimensionality-reducing randomized sketching, GreenMix's approach is effective when leveraging traditional stochastic descent to achieve its novel goals – leading to significant positive gains over the prior art (Sec. 7).

GreenMix Evaluation. *GreenMix's design is validated on a 15-node cluster of ARM big.LITTLE processors (Odroid N2+ testbed with Cortex-A73 and Cortex-A53 cores).* The evaluation is driven by a production-system function trace from Microsoft Azure [55] and serverless benchmarks from ServerlessBench [67] and SeBS [17] benchmark suites. GreenMix is also evaluated on an Intel Alder Lake 12^{th} generation i7-12700 big.LITTLE platform with 8 PCores and 4 ECores, and a simulation of 1000-node scale.

Our evaluation demonstrates that GreenMix is effective at saving energy and keep-alive cost while respecting the QoS target. GreenMix is close to both energy-optimal and keep-alive-cost-optimal strategies (within 10% of optimality). In contrast, state-of-the-art non-serverless asymmetric CMP techniques and serverless techniques (Linux Energy Aware Scheduler (EAS) [39], Octopus-Man [46], Hipster [44], IceBreaker [54], and HyHist [55]) are over 40% far from optimal strategies on two real-system-based and a simulation-driven evaluation. We also demonstrate that some design elements of GreenMix can be integrated into existing serverless techniques (HyHist [55] and IceBreaker [54]) in a complementary spirit to enhance their effectiveness on asymmetric CMPs.

Open-source artifact. GreenMix is available at the following link https://doi.org/10.5281/zenodo.10967591.

Table 1: Comparison of serverless works with GreenMix.

Schemes	Cold Start Mitigation	KAC	Adaptive	QoS	Energy	Asymmetric CMP
FaasCache [22]	✔	✔	✘	✘	✘	✘
IceBreaker [54]	✔	✔	✔	✘	✘	✘
HyHist [55]	✔	✔	✔	✘	✘	✘
Sequoia [61]	✔	✘	✔	✔	✘	✘
GreenMix	✔	✔	✔	✔	✔	✔

2 Related Works

Serverless research has tackled cold-start latency [5, 14, 45], stateful execution [9, 41, 48, 56], function chaining and data transfer [15, 28, 29, 40, 69], ML inference [7, 32, 57, 68], and performance isolation [19, 34, 36].

Table 1 lists recent works in the serverless domain that target a subset of GreenMix's goals. Here, *adaptive* refers to if the technique is adaptive to function updates and input changes. Shahrad et al. [55] (HyHist) is the first work to propose a simple yet effective auto-regression-based technique (hybrid histogram) to predict serverless function invocations that is also effective in industrial production settings. FaasCache [22] is a function caching technique that uses memory consumption and function re-invocation information to estimate functions to keep alive. IceBreaker [54] schedules functions on high-end and low-end servers following an FFT-based function invocation prediction scheme, to optimize for service time and keep-alive cost (KAC). Sequoia [61] uses several policies like function chain prioritization for warm starts, resource-aware scheduling, and reactive concurrency allocation to maintain QoS. To the best of our knowledge, GreenMix is the first work to demonstrate both energy and keep-alive cost savings for serverless workloads using asymmetric CMPs. Since the scope and goals of prior work were different, as expected, GreenMix is more effective than IceBreaker [54] and HyHist [55], but we also show that some design elements from GreenMix can be leveraged to enhance the effectiveness of HyHist and IceBreaker on asymmetric CMPs.

Work on asymmetric CMPs like Octopus-man [46] and Hipster [44] targets QoS for stateful microservices, but lacks support for serverless-specific aspects like cold starts and KAC. Octopus-man uses feedback control to map latency-critical tiers to big cores to satisfy SLOs, assuming long-running processes. Hipster uses reinforcement learning for tail-latency–aware placement in persistent microservice deployments. Linux Energy-Aware Scheduling (EAS) [39] places tasks on big.LITTLE cores using PELT-based utilization and an energy model to minimize system energy, but it operates at the per-task level and is agnostic to serverless lifecycles (e.g., cold-start probability, warm-pool sizing, and keep-alive policy). None of these solutions account for cold-start dynamics or KAC that dominate serverless platforms, motivating GreenMix's function-aware placement and keep-alive co-optimization on asymmetric multi-cores. We include them in our evaluation (Sec. 7).

3 Motivation

In this section, we describe the insights that enable GreenMix and the challenges in exploiting these insights (the experimental setup and methodology details are provided in Sec. 6).

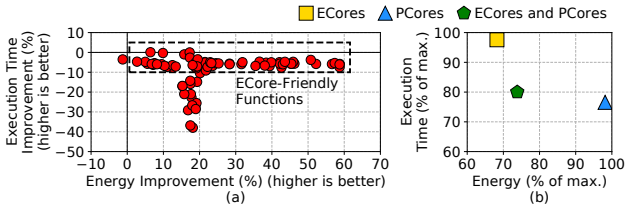


Figure 1: (a) Serverless offers the opportunity for a performance-energy trade-off. The improvement is expressed in terms of ECore execution compared to PCore execution. (b) Scheduling on heterogeneous CMPs is more effective.

Observation 1. *Many serverless functions provide an opportunity to save energy with minimal impact on their performance when executing on* ECores. *Executing these functions on asymmetric CMPs helps to exploit the performance-energy trade-off.*

Fig. 1(a) shows the execution time and energy savings for all ServerlessBench [67] and SeBS [17] benchmarks when they are executed on ECores vs. PCores. First, we observe that for a wide range of serverless functions, executing on ECore provides significant energy savings (up to 60% on average) compared to execution on PCores. However, the magnitude of performance degradation and relative energy savings are significantly different for functions. Therefore, a core-asymmetry-aware scheme needs to navigate this trade-off carefully. In fact, Fig. 1(b) demonstrates the higher effectiveness of using a mix of performance and efficiency cores, compared to a homogeneous/symmetric CMP. For fairness, we ensured that the homogeneous/symmetric setups had an equal number of cores (6 PCores or ECores) as the asymmetric setup (4 PCores and 2 ECores) and the same amount of memory (details in methodology Sec. 6). Asymmetric processors can provide over 20% energy savings on average while incurring less than 5% execution-time degradation compared to PCores-only setup.

While these results are encouraging, the serverless computing model has a unique characteristic of keeping functions alive in memory that incurs a keep-alive cost (KAC). Keeping function alive is effective for meeting the Quality of Service (QoS) latency target since the keep-alive strategy removes cold-start overhead - i.e., a warm-start helps us provide better QoS (lower service time). We note that the service time includes network RTT, control-plane overhead, execution time (varies depending upon PCore or ECore execution), and cold-start overhead (when the function execution is not warm started because it was not alive in memory at invocation time). Therefore, the QoS latency target can be expressed as the increase in service time of a function compared to a warm start on a PCore, which is the fastest service time a function can achieve.

We note that simply dividing the total execution occurrences on PCore and ECore in a weighted manner can not achieve the QoS target because of the cold start overheads which are difficult to mitigate due to unpredictability in function arrival pattern. Furthermore, the weighted approach is unaware of which functions provide higher energy-saving potential, leading to lost energy-saving opportunities. In fact, with cold starts, even when all executions are performed on PCore, it may not meet the QoS target because of the cold start overhead. Cold starts can be mitigated by keeping functions warm in the memory, but this incurs high keep-alive cost – highlighting the complex interplay between QoS, KAC, and

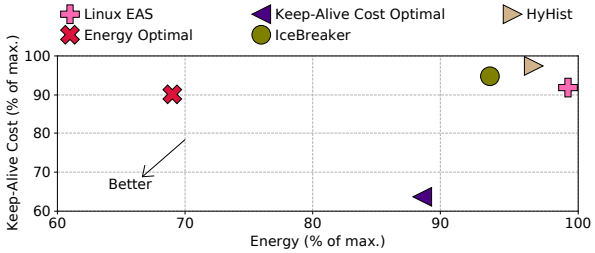


Figure 2: Joint optimization of keep-alive cost (KAC) and energy under a QoS target is challenging. The KAC and energy are normalized to the Linux EAS policy. Sec. 7 also shows that both non-serverless asymmetric CMP techniques and serverless techniques (Octopus-Man [46], Hipster [44], IceBreaker [54], and HyHist [55]) are suboptimal.

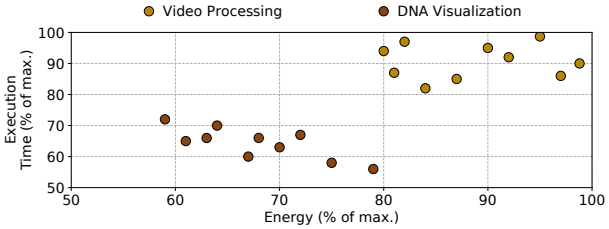


Figure 3: The energy consumption and execution time of functions varies with inputs and parameters.

energy savings. Therefore, it is important to understand if asymmetric processors can provide operational advantages in serverless settings (i.e., saving keep-alive cost and energy, while meeting a specified QoS latency target).

Observation 2. *Asymmetric processors cannot be trivially leveraged to obtain a desirable trade-off between energy and keep-alive cost using existing asymmetry-aware state-of-practice scheduling policies and serverless strategies.*

We perform a simple experiment where all serverless functions are executed on a real-system asymmetric multi-core processor (Odroid-N2+ big.LITTLE platform), following the invocation pattern of production-system trace [55] and using a fixed kept-alive policy used in production clusters [1, 2] (Sec. 6 details experimental methodology; for simplicity, the QoS target is configured as 20% increase in service time of a function compared to a warm start on a PCore, but trends persist across different thresholds, and in Sec. 7 we show that GreenMix's benefits persist across different configurable QoS targets). The goal of this particular experiment is to quantify the impact of scheduling policies on energy and KAC while meeting the QoS target.

Fig. 2 show multiple policies including Linux-EAS [39], Keep-Alive Cost Optimal, Energy Optimal, and two recent serverless strategies (IceBreaker [54] and HyHist [55]). Linux-EAS is the state-of-the-practice Linux energy-aware scheduling policy that performs energy-aware task placement on asymmetric CMPs. The Keep-Alive Cost Optimal and Energy Optimal policies are obtained by an exhaustive search over all possible combinations of functions and cores to find the minimal KAC and energy, respectively. Keep-Alive Cost Optimal and Energy Optimal policies are practically infeasible due to the use of future knowledge.

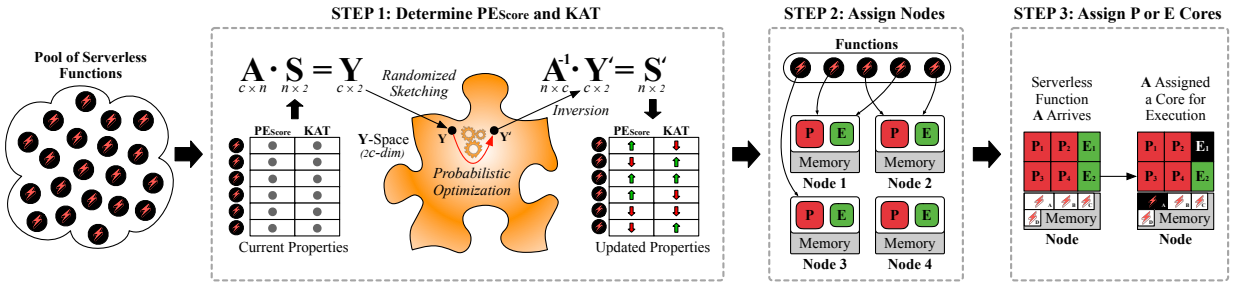


Figure 4: GreenMix's three-step methodology: (1) determine the PE_{Score} and keep-alive time (KAT) using randomized sketching and inversions, (2) assign nodes to each function for warming up, and (3) assign a `P` or an `E` core to a function upon invocation.

The Keep-Alive Cost Optimal policy saves over 30% KAC compared to Linux-EAS but is sub-optimal in terms of energy. Energy Optimal provides significant energy savings (over 30%), but at the expense of much higher KAC compared to the Keep-Alive Cost Optimal policy. As expected, recent serverless strategies are sub-optimal, asymmetry-unaware, and do not focus on co-optimizing energy and KAC. These trends indicate the opportunity of designing a new energy- and KAC-aware scheduling policy that can exploit the energy and KAC trade-offs while meeting the QoS targets.

Exploiting this trade-off space of KAC and energy is challenging because of the irregular invocation pattern of functions, varying performance, energy consumption characteristics among functions, and change in characteristics across different invocations of the same function, as shown next.

Observation 3. *Not all invocations of the same serverless function are the same. While there is some predictability, different invocations may exhibit different* `PCore` *and* `ECore` *affinity due to differences in inputs/parameters.*

As an example, we track the execution characteristics of two particular functions during different invocations: video processing and DNA visualization in the SeBS benchmark [17]. Fig. 3 shows their invocations on `PCores` only, and observes that even when executing on the same core type, both functions show significant variation in terms of their execution time and energy consumption across different invocations – this is because different invocations may use different function parameters or inputs. Consequently, it affects how a strategy should prioritize a function in terms of warm up (warm up priority) and execution (`PCore` execution priority/core affinity). This observation emphasizes the need for handling dynamic function invocation behavior. We also highlight that an effective design needs to consider the co-optimization of different functions in the pool of simultaneously invoked functions because different functions provide different opportunities for energy savings and opportunities for reduction in keep-alive cost due to the differences in their future invocations probabilities and their memory sizes.

Finally, we also note that it is critical to jointly optimize keep-alive cost (KAC) and energy. If only the KAC is optimized under a QoS target, then the functions will tend to have less keep-alive period, which will result in more cold starts and more `PCore` executions (to meet the QoS), thus, increasing the energy consumption. Similarly, if only energy saving is optimized under a QoS target, then functions will be prioritized to be executed on `ECores`, which will violate QoS target and force functions to increase keep-alive time to minimize the cold start overhead in an attempt to meet the QoS, but result in higher keep-alive cost. GreenMix's design considers these challenges and opportunities (Sec. 4), and its evaluation demonstrates its effectiveness (Sec. 7).

4 The Design of GreenMix

In this section, we present the design of GreenMix to leverage `PCores` and `ECores` for serverless execution. GreenMix's goal is to minimize the overall energy consumption and keep-alive cost while meeting the QoS target.

Overview of GreenMix. GreenMix has three major design steps (Fig. 4). First, GreenMix takes functions that are candidates for being warmed up and estimates their relative PE_{Score} and keep-alive time (KAT) (Sec. 4.1). The PE_{Score} determines a function's likelihood of being scheduled on a `PCore` vs. an `ECore` – essentially, PE_{Score} acts as an indirect proxy for a function's performance and energy core friendliness (opportunity scope for performance-energy trade-off). The keep-alive time (KAT) determines how long the function should be kept warm in memory. GreenMix employs randomized sketching and inversions [37, 47] to perform these function score estimations in the presence of a large number of functions and their varying behaviors (e.g., input parameters). Randomized sketching compresses the high-dimensional function features into a much smaller random subspace (e.g., Johnson–Lindenstrauss projections) that approximately preserves distances and inner products. GreenMix estimates PE_{Score} and KAT using this compact sketch rather than the full data, achieving near-linear runtime and low memory while maintaining accuracy for robust ranking across thousands of functions and input variants.

Second, GreenMix determines the node to warm up each serverless function and place each cold-started serverless function. These design trade-offs and decisions are discussed in Sec. 4.2. Finally, when a function is invoked (either warmed-up or cold-started), GreenMix needs to select the most appropriate core type on the assigned node. Sec. 4.3 discusses the challenges and solutions for choosing the effective core type to meet GreenMix's objectives.

4.1 STEP 1: Determine the PE_{Score} and KAT

As a first step, GreenMix estimates two properties (PE_{Score} and KAT) for all warmup candidate functions and optimizes them periodically to achieve near-optimal energy and keep-alive cost saving while respecting QoS target. Unfortunately, accurately determining these scores for all warmup-eligible functions is challenging because of

the large number of functions active in production settings [55, 63]. This is why GreenMix leverages the theory of randomized sketching to *probabilistically optimize* these parameters for a large number of functions by reducing the dimensionality of the large space and producing near-optimal results despite information loss. Next, we provide the definitions of PE_{Score} and KAT, before demonstrating how they are estimated and optimized.

The PE_{Score}. This score (a value between 0 and 1) indicates a function's likelihood of being scheduled on a `PCore` vs. an `ECore`. The PE_{Score} essentially attempts to capture the net benefit of running on an `ECore`. It is high for a function that observes a large energy reduction when run on an `ECore` without significant degradation in performance. If this metric is high, it indicates large energy savings for small performance degradation, i.e., the function is `ECore`-friendly (a proxy for improving energy savings). A low PE_{Score} indicates that the function is `PCore`-friendly. However, the PE_{Score} is not necessarily required to be accurately measured every time for every function (estimations can be effective in practice). The randomized sketching enables approximate estimations of PE_{Score} that GreenMix leverages. GreenMix's randomized sketching also probabilistically updates these estimations and optimizes them over time, such that using PE_{Score} as a likelihood for being scheduled on a `PCore` vs. an `ECore` leads to energy savings and QoS satisfaction.

The Keep-Alive Time. KAT is the time that the function should be kept alive (warmed up) in node memory after it has finished its execution to ensure that it receives a warm start, should it be invoked again in the near future. This property is directly proportional to the keep-alive cost – the lower the keep-alive time, the lower the keep-alive cost – and *therefore, the KAT can be used as a direct proxy when optimizing for keep-alive cost.* Therefore, *optimizing the PE_{Score} helps GreenMix maximize the energy savings, and optimizing the KAT helps GreenMix minimize the keep-alive cost.*

Note, both PE_{Score} and KAT values are chosen arbitrarily when a function is invoked for the first time; they are then updated and optimized over future invocations periodically. The optimization interval for updating the parameters is configurable and off the critical path. The interval is set to one minute because GreenMix's randomized sketching and inversion finishes in less than a second.

Randomized Sketching for Optimizing PE_{Score} and KAT

Why Randomized Sketching for GreenMix? Optimizing the scheduling decisions based on PE_{Score} and KAT for a large number of functions is challenging because of the high dimensionality and dynamic nature of the environment. Traditional optimization methods like gradient descent require maintaining estimated PE_{Score} and KAT for each function and co-optimizing all functions together, resulting in limited solution effectiveness in the presence of a large and frequently changing mix of serverless functions. Outlier behavior and noisy characteristics are also known to compromise overall solution quality in gradient descent methods [25, 30]. Alternatives such as principal component analysis suffer from large decision-making overhead to calculate the eigenvectors of the covariance matrix [65] – impractical in an online setting.

To mitigate this challenge, GreenMix leverages the concept of randomized sketching [37, 47], which has been shown to be effective in optimization problems with large dimensionality in various

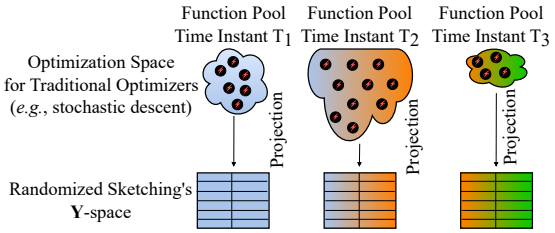


Figure 5: Traditional optimizers suffer due to a large and frequently changing function mix. Randomized sketching embeds the properties of function pools in a lower-dimension space without requiring one-to-one function correspondence, with a configurable loss parameter.

domains (visualized in Fig. 5) [4, 13, 18, 20, 23, 27]. Randomized sketching projects a large-dimensional space to a small-dimensional space, and the parameters of interest can be optimized in the lower-dimensional space and projected back to the original space – enabling more efficient co-optimization and achieving robustness against noise. There is some expected information loss and suboptimality due to embedding of information in smaller space, but prior works have shown that empirically it performs competitively with reasonable theoretical guarantees [12, 21, 31]. This information loss can be controlled via a configurable parameter, and our evaluation confirms that GreenMix is effective across different values of configurable loss parameter (Sec. 7). Experimentally (not shown for brevity), we found that GreenMix's randomized sketching indeed yields higher energy and keep-alive cost savings than other alternatives such as stochastic descent (GreenMix is 10% within optimality compared to stochastic descent being over 20% far from optimality). Next, we describe how GreenMix utilizes randomized sketching.

How does Randomized Sketching Work for GreenMix? As shown in Fig. 4, GreenMix uses randomized sketching to project samples from an n-dimensional space to a c-dimensional space; $c << n$. In GreenMix's case, n corresponds to the number of functions, and each function has two attributes: the PE_{Score} and the KAT value. Let S be an $n \times 2$ matrix that represents the tableau of PE_{Score} and KAT values of all n functions (functions in rows, PE_{Score} in the first column, and KAT in the second column). We want to project S onto Y, which is a $c \times 2$ matrix. Note that the c-dimensional space is not trivially c functions as a subset of the original large number of n functions; it is a transformation that compresses the information to the lower dimension.

Randomized sketching achieves this information compression using a matrix A, which is $c \times n$, such that $Y = A \cdot S$. Note that randomized sketching provides the convenience to choose A independent of the contents of the samples (PE_{Score} or KAT). As a theoretical property of randomized sketching, matrix A and the value of c control the loss of information when going from S to Y [8, 33]. The Johnson-Lindenstrauss Lemma [35] guarantees that if the elements of A are randomly sampled from a Gaussian distribution (as is the case with GreenMix) and c is selected such that $c \geq \frac{log(n)}{\epsilon^2}$, where ϵ dictates the tolerance of information loss, the dimensionality of the problem can be reduced from n to c without much loss of information [12]. The lemma guarantees that the distance between all samples is preserved up to a factor of $\pm\epsilon$.

GreenMix is configured to keep the information loss to be below 10% (typically used setting [6]), and our evaluation also performs a sensitivity study against ϵ parameter for robustness (Sec. 7). It sets the corresponding c to a constant value across different optimization intervals, although n (the number of functions) can freely vary with load fluctuations without restriction. GreenMix divides the function set into multiple sets and runs multiple optimizers independently when the value of n is large enough to cross the information loss threshold ϵ. The next step is to leverage this compressed information for optimizing PE_{Score} and KAT.

Stochastic Optimization of PE_{Score}, KAT in Low-dimensional Y-Space. The Y-space is a $2c$-dimensional space, where at each optimization interval, GreenMix receives one point in this $2c$-dimensional space because each step produces one Y matrix. There are a large number of possible Y matrices in this space, each corresponding to different PE_{Score} and KAT values that need to be optimized. We now describe the process of finding the optimal PE_{Score} and KAT values in the low-dimensional space.

As shown in Fig. 4, GreenMix treats the problem space as a Gaussian process and uses an acquisition method to sample points (Y matrix) in the problem space: $Y \rightarrow Y'$. The idea behind this optimization is to steadily optimize the PE_{Score} and KAT values of functions, interval after interval, until they reach their near-optimal values for a certain objective. The optimal choice of KAT affects the warm start rate of functions, which plays a key role in maintaining a function's QoS. The optimal choice of PE_{Score} affects a function's energy consumption.

GreenMix's objective is to minimize the aggregate energy consumption and keep-alive cost of all functions with equal weight placed on both (the weights can be configured). If the QoS requirement is not met, then the objective receives a value of infinity. Thus, the optimizer steers away from such points. GreenMix uses the "expected improvement" acquisition method to sample the next point (Y') in the vicinity of the current one based on its prediction of which point is likely to provide the highest expected improvement over the objective value of the current point. The prediction is made based on the Gaussian process that is modeled over the entire problem space based on already sampled points. As more and more points are sampled in the Y-space, the prediction quality improves and the objective is optimized. Overall, GreenMix optimizes over multiple samples under varying conditions.

Next, we make two important design choice remarks about how GreenMix handles the dynamics of functions and their input parameters changing over time. GreenMix's robustness in these aspects are experimentally substantiated in Sec. 7.

GreenMix optimizes in the vicinity of the current Y values so that the new Y' value is not too different from Y, i.e., the new PE_{Score} and KAT values of functions are not too different from their previous values. This improves stability and avoids oscillations. As a function's behavior changes over time, GreenMix automatically adjusts to the new behavior by again optimizing the PE_{Score} and KAT values (Sec. 7). The values are continuously optimized.

The function set may be different for every interval (Y represents different functions for every interval, even though its dimensions are constant). The same function (or ones with similar properties) maps to the same areas in a few of the $2c$ dimensions, while other functions map to other areas on other dimension axes. Thus, the GreenMix optimizer can optimize across different invocations of a function, even if it misses a few intervals, as the Y mapping brings it back to where it left off and optimizes from that point onward. This enables adaptation to varying function behavior (Sec. 7).

Algorithm 1 Pseudocode for function core selection.

```
Input: f ← Function that requires execution
if QoS is estimated to be met in ECore execution then
    if multiple ECores are available then Execute f on one ECore
    else Execute f on the ECore with probability PE_Score(f)
else if QoS is not estimated to be met in ECore execution then
    if multiple PCores are available then Execute f on one PCore
    else Execute f on the PCore with probability 1−PE_Score(f)
```

Next, we discuss how GreenMix re-projects the optimized PE_{Score} and KAT values to the original space for all functions.

Inversion of Randomized Sketch to Obtain the Optimized PE_{Score} and KAT Values. As shown in Fig. 4, once GreenMix has the Y' matrix, the next step is to get to the S' matrix, which gives us the PE_{Score} and KAT values for all n functions. To get to S', GreenMix invert's the earlier process of calculating Y: $Y = A \cdot S \Rightarrow S' = A^{-1} \cdot Y'$. Note that A^{-1} is the inverse of the earlier A matrix with elements sampled from a Gaussian distribution. GreenMix leverages the diverse pool of pre-generated and inverted A matrices to speed up. This is because the A matrix is independent of serverless function characteristics, and generating it does not fall on the critical path of function execution. The S' matrix directly outputs the optimized PE_{Score} and KAT values for all the functions eligible for warm up. Next, we see function node assignment.

4.2 STEP 2: Placement of Functions on Nodes

The next step is to map the set of all placement-candidate functions to the set of nodes. GreenMix's decision process for placement-candidate functions is categorized: *(a) warmup-eligible functions:* these functions are ready to be warmed up in one of the nodes following the warm up prediction scheme; *(b) invoked functions:* these functions have been invoked without a warm up (i.e., they undergo cold-start). Functions are warmed up for an amount of time decided by stochastic optimization, as was discussed in Sec. 4.1.

The key insight behind GreenMix's function placement is to balance two competing trade-offs: *(a) maximize the PE_{Score} diversity of functions on each node.*, and *(b) adapt to the available memory capacity in each node.* A higher PE_{Score} diversity increases the probability of utilizing both performance and efficiency cores. If GreenMix did not manage this aspect carefully, a node with multiple `PCores` and `ECores` may end up warming up `PCore`-friendly functions only, losing out on the energy improvement opportunity of `ECores`.

First, GreenMix maintains a `PCore`/`ECore` ratio as a score for each node; this score attempts that all nodes have the same ratio of `PCore` and `ECore`-friendly functions, which is as close as possible to the ratio of `PCores` and `ECores` on a node. This ensures a desirable diversity of function types, which increases the likelihood of a function being run on a core that it has the most affinity towards when it is invoked. Second, GreenMix opportunistically adapts to the available memory capacity in each node. Maximizing the PE_{Score} diversity of functions on each node can be sometimes conflicted with available memory capacity on the node, since the

memory requirements of functions with different PCore/ECore ratios may not always perfectly add up to the available memory on the node. While this placement decision is not on the critical path if the function is being warmed up (i.e., not being cold-started), the optimization space must be manageable, especially when the number of functions is large.

One important GreenMix-specific consideration is the affinity of serverless functions to previously assigned nodes. GreenMix ensures that mapping assignments are not random when correlations among functions are specified. Note that the functions that suffer cold-start are placed quickly because the node placement decision-making falls on the critical path of their execution. Therefore, GreenMix places them on nodes with available memory and cores as quickly as possible to minimize control plane overhead. Also, when GreenMix places a serverless function, it does not reserve the cores on the node; however, the memory reserved for a serverless function is reserved for the full keep-alive time. As needed, this memory can be relinquished by the functions.

4.3 STEP 3: Core-type Selection for Functions

The placement process described in Step 2 determines the node location where a particular function is warmed up or cold started. Step 3 selects the core type (PCore vs. ECore) that should execute a given function upon invocation. GreenMix recognizes that when a function is invoked, the mix of other functions on the node may be different from when it was placed on the node. Thus, when a function is invoked, GreenMix needs to make a fresh decision about which core type should be used for execution.

Trivially selecting the core type based on a function's PE_{Score} is not effective, and may not even be possible. First, such a one-dimensional strategy would unnecessarily hurt the performance of ECore-friendly functions (high PE_{Score}) in an attempt to maximize the energy savings of the system. Further, a node may potentially host more functions than the number of available cores of a particular type, and all core types may not be available at all times, depending on the current occupancy of the node. *GreenMix's core type selection process for execution balances competing trade-offs: (a) QoS target of the invoked function, (b) performance/efficiency-core affinity of the function, and (c) availability of different core types on the node upon invocation.*

Algorithm 1 summarizes GreenMix's core type selection process. GreenMix uses function execution time and energy on PCores, ECores, memory footprint, and cold-start overhead as parameters to recognize the best processor. GreenMix's first consideration is the QoS requirement of the invoked function. GreenMix tracks a function's historical execution times to estimate whether it is meeting its QoS target – this tracking is not specific to GreenMix; all schemes need to track the QoS target. GreenMix leverages this historical execution time tracking for core-type selection.

If a function's previous execution history indicates that it will meet its QoS if executed on ECore this time (Line 2), possibly due to previous many executions on PCore or warm starts, then, GreenMix aims to schedule it on an ECore to improve the overall energy savings. If multiple ECores are available, then GreenMix schedules it on an ECore (Line 3). However, if only one ECore is available, then GreenMix probabilistically schedules it on an ECore with the probability being equal to its PE_{Score} (Line 4). Recall that the higher the PE_{Score}, the more ECore-friendly the function is, and the higher the energy savings obtained from running that function on an ECore. This probabilistic approach allows to ensure that the only remaining ECore is not wasted on a function with a low PE_{Score} if a function with a relatively higher PE_{Score} is invoked right after – the newly invoked function would be scheduled on a PCore, sacrificing its energy savings potential.

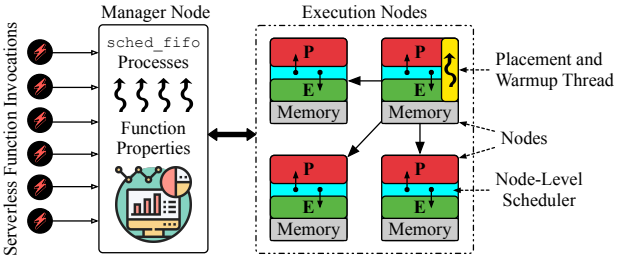


Figure 6: Implementation overview of GreenMix's design.

Similarly, the functions that cannot meet their QoS on an ECore are scheduled on PCores (Line 5). Here, if multiple PCores are available, then the function is executed on a PCore (Line 6). Otherwise, it is probabilistically scheduled on a PCore with probability 1 - PE_{Score} (Line 7). This ensures that the lower the PE_{Score}, the higher the job's probability of being scheduled on a PCore, to ensure similar but inverted logic as with ECore scheduling. Overall, Algorithm 1 opts for a simple decision-making logic for core type selection as the selection decision is on the critical path of function execution to make control plane overhead negligible (Sec. 7).

Overall, GreenMix recognizes that it is possible that a small fraction of functions may not meet their QoS target at all times. This is because of the unavailability of PCores due to load level changes, input parameter changes, and the need to execute the function on both PCore and ECore when changes are detected. But, as our evaluation shows (Sec. 7), the impact is limited in practice. Also, GreenMix adapts to available resources and changing load levels (i.e., dynamic environment with changing non-serverless job load), and does not assume that all core types on a node are available for executing an invoked serverless function (Sec. 7).

5 GreenMix Implementation

GreenMix is implemented as a (1) system-level dispatcher and a (2) node-level scheduler (Fig. 6). Upon function invocation, (a) its Docker image is fetched from remote storage, (b) an in-memory container is created from it, and (c) the function is executed.

System-level dispatcher. This dispatcher is implemented in two levels. The first level runs on the manager node, and it dispatches serverless functions to the execution nodes. This node runs a set of `sched_fifo` processes, implemented via `Python Multiprocessing`. These processes listen to an interrupt handler, which is invoked with a `sched_yield()` system call upon the invocation of a function. Python `multiprocessing_pool` class is used to dynamically control the number of processes on the manager node depending on the current load of function invocation. The second level of the dispatcher runs on all the execution nodes. This second-level dispatcher listens for function invocation interrupts while running

serverless functions. Lightweight threads, created by `clone()` system calls, are used to listen for these interrupts. These threads share the same *Group ID*, allowing the kernel to share resources like address space, physical memory pages, signal handlers, and files among the different threads.

Node level scheduler. When a function is placed on a node, the node-level scheduler decides which type of core to run the function on. It uses `sched_setaffinity()` system call of the Linux *Heterogeneous Multicore Processing (HMP)* module to bind a function for execution on a specific core type. After the execution of a function on a node, its execution time, cold start time, and memory consumption information are passed to the manager node using Linux `secure copy`. This information gets stored in the historical hybrid database (OrientDB), maintained at the manager node. GreenMix does not modify the kernel, but its dispatcher and scheduler make decisions that override native kernel decisions. *All of these control plane overheads are included in our results (Sec. 7).*

Operational Considerations. Much of the heterogeneity overhead is handled transparently in hardware: ARM big.LITTLE maintains cache coherency via AMBA CHI, and Intel Alder Lake's Thread Director manages core migration and power-state transitions. For analysis and debugging, we aggregate Linux perf counters by CPU masks (e.g., PCores 0–3, ECores 4–5) and add a single `core_type` attribute to serverless logs. Users remain insulated by the serverless abstraction. From an operator's perspective, GreenMix can be integrated with existing serverless frameworks (e.g., Apache OpenWhisk [1]) by delegating placement decisions to GreenMix through an interface while retaining the platform's container/runtime pipeline. All control plane overheads are included in our results (Sec. 7).

6 Experimental Methodology

Experimental Setup. We evaluate GreenMix on a prototype cluster consisting of 15 Odroid N2+ nodes [3]. Each node has 4GB of memory and an Amlogic S922X big.LITTLE processor with 4 ARM Cortex-A73 cores (`PCore`) and 2 ARM Cortex-A53 cores (`ECore`). Cortex-A73 runs at 2.2 GHz and draws 1.5-2.8 watts per core. Cortex-A53 runs at 1.9 GHz and draws 0.6-1.2 watts per core under typical serverless workloads. We added a 128GB microSD card and installed Ubuntu MATE 20.04 for each node. A picture of our cluster setup is shown in Fig. 7. We use the Likwid Powermeter [62] to extract both the core and DRAM energy consumption data from the hardware register interface. To ensure consistent measurements, automatic DVFS was not activated (to avoid unpredictable variations).

To evaluate GreenMix's effectiveness on different types of asymmetric CMPs, we also evaluated GreenMix on one Intel Alder Lake 12^{th} generation i7-12700 big.LITTLE processor (launched in Nov. 2021). It has 8 golden cove performance core (`PCore`) and 4 gracemont efficient cores (`ECore`) [50, 51]. In Alder Lake, the `PCores` consume approximately 5 - 8 W per core (under Turbo), while `ECores` draw 1.5 - 3 W. Total socket power can scale up to 125 W.

To evaluate GreenMix's robustness on a large-scale system, we developed a simulation platform with 1000 nodes to support the execution of 8.75 million functions, using the data collected from the prototype cluster. In Sec. 7, we show that our simulation results also confirm GreenMix's effectiveness.

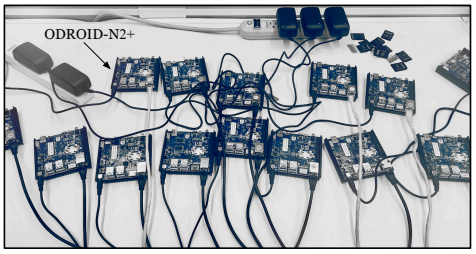


Figure 7: The prototype setup to evaluate GreenMix.

Serverless Workloads. GreenMix's evaluation is driven by real-world applications and industry traces. The functions are generated from the ServerlessBench [67] and SeBS [17] benchmark suites. These are widely-used serverless benchmark suites [42, 66, 71]. These benchmarks cover a wide range of real-world domains, including image processing, machine learning inference, data analytics, online compiling, and linear algebra. They also cover a variety of cold start times and I/O overhead characteristics. To represent production-like workload arrival patterns and characteristics, we use Microsoft Azure Function trace [55]. Our production-grade trace used for evaluation consists of serverless invocation data for two weeks on Microsoft production systems. This trace includes the inter-arrival time of functions, the memory allocated for the function, and its execution time. GreenMix follows the same inter-arrival pattern of functions as the Azure trace. For faithful representation, from the memory allocated and execution time information, we pick the closest matching benchmark from our benchmark pool to represent the corresponding function behavior in the trace.

Relevant Solutions. We compare against multiple asymmetric core-aware "non-serverless" schemes which are similar in spirit (asymmetric CMP, energy consumption, QoS): **Linux Energy Aware Scheduler (EAS) [39]** extends the Linux kernel to make it fully aware of the power/performance capabilities of asymmetric CMPs, to optimize energy consumption for advanced multi-core SoCs including big.LITTLE. **Octopus-Man [46]** is a latency and QoS-aware task placement strategy on asymmetric CMPs. It uses feedback control mechanisms to handle uncertainties in workload fluctuations and the runtime dynamics of a real system. **Hipster [44]** uses ARM's big.LITTLE platform to improve resource efficiency and energy consumption, by deciding their core placement using several heuristics and a reinforcement learning model.

IceBreaker [54] and HyHist [55]. These serverless strategies do not attempt to exploit asymmetric CMPs to optimize energy and keep-alive cost savings. Nevertheless, we enhance IceBreaker and HyHist via augmenting their warm up and keep-alive time strategies for reducing keep-alive cost with GreenMix's placement and core-selection strategies (step 2 and 3) based on estimated `PCore`/`ECore` friendliness to save energy while meeting the QoS constraint. We refer to these enhanced-by-GreenMix techniques as **IceBreaker+GM** and **HyHist+GM**. *The purpose of this modification is to demonstrate that GreenMix is complementary (not only competitive), and partially leveraging some design elements of GreenMix enhances the effectiveness of existing serverless schedulers.*

Energy-Opt and **KAC-Opt.** We designed these two techniques to employ a brute-search approach to determine the optimal function warm up, placement, and execution decisions that minimize

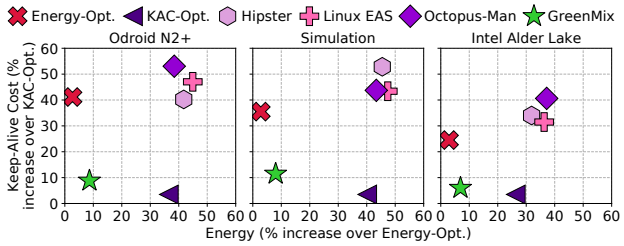


Figure 8: GreenMix achieves near-optimal energy consumption and keep-alive cost, while meeting the QoS.

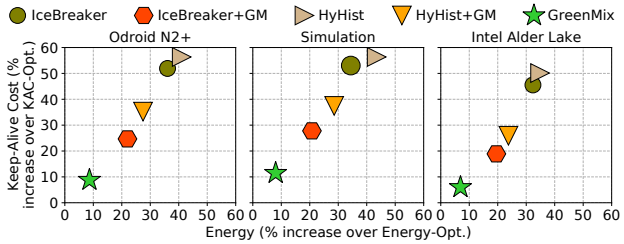


Figure 9: GreenMix optimizes keep-alive cost & energy consumption, & meets the QoS, compared to state-of-the-art.

energy (Energy-Opt.) or KAC (KAC-Opt.). Energy-Opt. only minimizes energy, but it is sub-optimal for KAC, and vice-versa. Energy-Opt and KAC-Opt are determined offline via brute-forcing different values of execution decisions and keep-alive time for each function at each time instant (instead of a fixed-value throughout).

Evaluation Metrics. The goal of GreenMix is to minimize the **energy consumption** and **keep-alive cost**, while meeting the **QoS** target. Energy consumption is considered for all components (compute and memory) to keep functions alive, place them, and execute them. Keep-alive cost (KAC) is the cost incurred for reserving memory to keep alive a function – expressed as the product of cost per unit memory per unit time for reserving a node and the keep-alive time of the function. QoS target is a configurable latency target defined as the increase in service time of a function compared to a warm start on a `PCore`, which is the fastest service time a function can achieve. The default threshold is set to be 20%, but is configurable. Sec. 7 also confirms that GreenMix's benefits persist across different QoS targets.

7 Evaluation and Analysis

GreenMix achieves the closest energy consumption to Energy-Opt and closest keep-alive cost to KAC-Opt compared to other techniques on average on different hardware and simulation platforms, while meeting QoS targets. For visual clarity, we present results separately for comparing with non-serverless asymmetric CMP techniques and serverless techniques (Fig. 8 and Fig. 9).

Fig. 8 and Fig. 9 show the energy consumption and keep-alive cost of GreenMix and competitive techniques on three platforms: Odroid N2+, Simulation, and Intel Alder Lake. GreenMix achieves a much lower energy and keep-alive cost than competitive techniques, and it achieves the best balance between Energy-Opt and KAC-Opt on their respective metrics. For example, on Intel Alder Lake, GreenMix has a less than 10% increase in energy and KAC than Energy-Opt and KAC-Opt, while competitive techniques have a greater than 40% increase in energy and KAC because non-serverless asymmetric CMP techniques are not aware of serverless-specific characteristics (cold-start, keep-alive period, etc.) and known bottlenecks of reinforcement and other learning algorithms in the presence of high dimensionality [23].

Table 2: QoS violations across all invocations.

Technique	% QoS Violation	Technique	% QoS Violation
GreenMix	0.12%	Octopus-Man	8.23%
Energy-Opt	0.13%	IceBreaker	9.23%
KAC-Opt	0.12%	Linux EAS	10.0%
Hipster	7.77%	HyHist	12.02%

Fig. 9 shows that IceBreaker and HyHist are less effective than GreenMix, which is expected because their optimization goals are different. In particular, we note that IceBreaker selectively warms up functions on "cheaper memory and slower compute nodes" to reduce keep-alive cost [54]. Unfortunately, this scenario does not apply to asymmetric processors because the functions do not have access to different types of memory to reduce keep-alive cost and need energy-saving mechanisms (selection of different core types) to save energy. This leads to IceBreaker's poor effectiveness on asymmetric CMPs. Interestingly, we show that some parts of GreenMix can be leveraged to enhance IceBreaker and HyHist by 20% (IceBreaker+GM and HyHist+GM) – via integrating GreenMix's placement and core-selection strategies (step 2 and 3) in IceBreaker and HyHist. In IceBreaker+GM and HyHist+GM, we keep the core of IceBreaker's and HyHist's warm up and keep-alive time strategies for reducing keep-alive cost (i.e., FFT-based prediction and hybrid-Histogram based prediction), and amplify their effectiveness on asymmetric CMPs via partial elements from GreenMix.

Next, Table 2 shows the percent of times the QoS targets (threshold of 20% service time increase over `PCore` warm start time for all functions) are violated for different techniques. While GreenMix only violates the QoS requirements ≈0.12% of the time, competitive techniques violate the QoS 7-13% of the times. This is because competing techniques either do not enforce QoS constraints (Hipster, IceBreaker, HyHist), or their QoS target constraints are more relaxed due to their context (Octopus-Man). This shows GreenMix can save energy and keep-alive costs while following QoS targets.

GreenMix's effectiveness is not concentrated on only a few functions. GreenMix outperforms for nearly all individual functions compared to other competitive techniques for energy, KAC, and QoS (Fig. 10). Note that the energy plot does not show Energy-Opt, and the KAC plot does not show KAC-Opt, as their line are at 0 in their respective plots. Going forward, we present all results for the Odroid N2+ devices for brevity; the observed results and trends remain consistent for Intel Alder Lake and simulation as well.

GreenMix's randomized sketching technique helps it make more optimal decisions for energy consumption and keep-alive cost while meeting QoS. To study why GreenMix performs well, we see how GreenMix performs with and without sketching.

When implemented without sketching, we maintain the same number of functions for all intervals so that the optimization space dimensions remain constant. The optimization space dimensions are much greater without sketching than the number of dimensions with sketching. The two heatmap plots in Fig. 11 compare how close GreenMix, with and without random sketching, comes to selecting

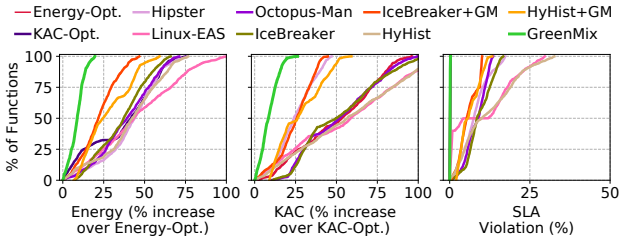


Figure 10: GreenMix provides benefits for all functions.

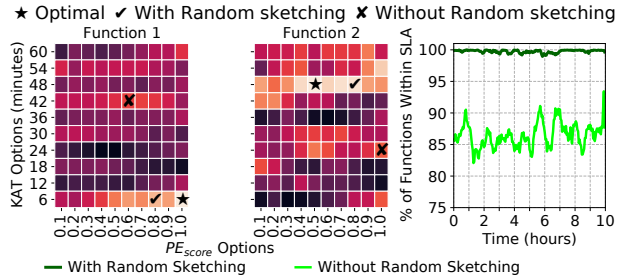


Figure 11: Randomized sketching in GreenMix improves the optimization and helps to maintain QoS.

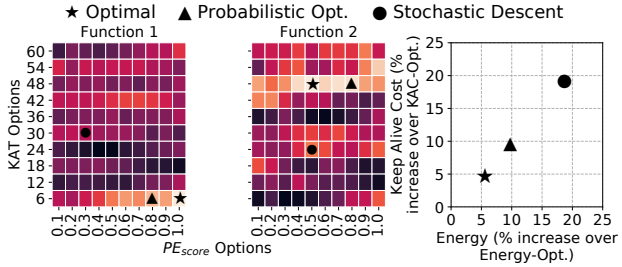


Figure 12: Probabilistic optimization in GreenMix helps it to converge to more optimal solutions than stochastic descent.

the optimal PE_{Score} and KAT configurations for two example functions. As the plots show, GreenMix's randomized sketching comes much closer to selecting the optimal configuration than when it is implemented without randomized sketching. As the third plot in Fig. 11 shows, without randomized sketching, the QoS requirement is also violated at a large rate (10-17%) most of the time. This is due to the fact that without random sketching, the search space becomes very large due to the high dimensionality of the Y-Space, which makes it difficult to reach the optimal solution quickly. Although this decision overhead is not on the critical path, it is critical to keep it as low as possible to allow frequent decision-making.

GreenMix's randomized sketching based design and probabilistic optimization to search for optimal PE_{Score} and KAT values help it make significantly better decisions than traditional approaches like stochastic descent. The heatmaps in Fig. 12 compare how closely GreenMix with probabilistic optimization and with stochastic descent approach the optimal PE_{Score} and KAT configurations for two sample functions. GreenMix's probabilistic strategy consistently comes closer to the optimal due to its ability to escape local optima and adapt to dynamic conditions, unlike stochastic descent, which struggles with high dimensionality.

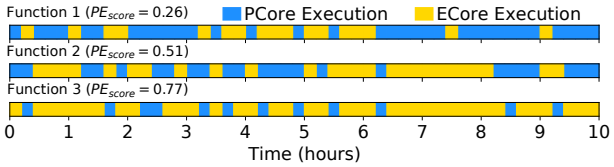


Figure 13: Core type selection of GreenMix varies across different invocations for functions with different PE_{Score}.

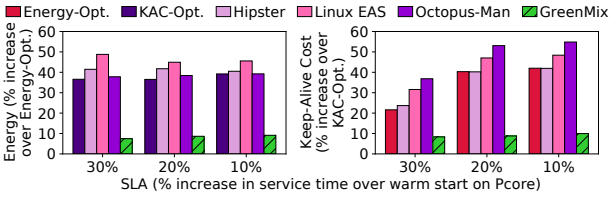


Figure 14: GreenMix adapts to different QoS targets.

Still, stochastic descent achieves results within 20% of optimality – worse than GreenMix's 10% but significantly better than prior techniques, which exceed 40% deviation. This shows that even with traditional optimization replacing randomized sketching, partial benefits of GreenMix can be realized.

GreenMix's PE_{Score} enables functions to be probabilistically scheduled on *both* types of cores instead of naively being skewed to only one favorable core-type. Fig. 13 shows the `PCore` and `ECore` executions of three functions with different PE_{Score} values over a span of 10 hours. Function 1 (average PE_{Score} of 0.26) has a high percentage of executions on `PCore`, but sprinkled with `ECore` executions all throughout the time span. Function 3 (PE_{Score} of 0.77) has a high percentage of executions on `ECore`, with `PCore` executions all throughout, and Function 2 has a behavior somewhere in between. The figure highlights two properties of GreenMix: (1) all functions have the opportunity to run on both core types due to GreenMix's probabilistic selection instead of being skewed to only one favorable type, and (2) GreenMix's ensures that `PCore` and `ECore` executions are spread out across time and not bunched together, which could cause QoS violations (if all `ECore` executions happened together).

GreenMix's adapts effectively to different QoS targets. Fig. 14 shows how GreenMix performs for QoS requirements of 30% and 10% service time degradation (in addition to the default 20% degradation threshold relative to their service time when executed on a `PCore` with a warm start). We note that the KAC-Opt and Energy-Opt are naturally different for each QoS target in Fig. 14. As the figure demonstrates, across all varieties of QoS requirements, GreenMix has the lowest energy consumption and keep-alive cost on average compared to all other techniques. GreenMix also maintains consistent energy and KAC performance for all QoS targets, while other techniques perform differently based on the QoS target.

GreenMix is effective across different ratios of `PCores` to `ECores` compared to corresponding optimal points. ***GreenMix also demonstrates that a heterogeneous system performs much better for serverless workloads compared to a homogeneous/symmetric system with just `PCores` or `ECores` only.*** Fig. 15(a) shows how GreenMix performs for different `PCore`:`ECore`

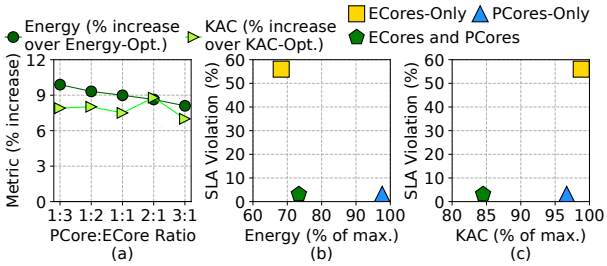


Figure 15: GreenMix improves key results over baseline for a wide range of `PCore:ECore` ratios. It provides benefits over homogeneous configurations of core types (symmetric CMP).

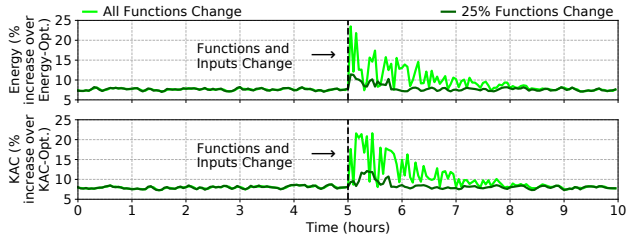


Figure 16: GreenMix adapts to input and function changes.

ratios. The energy (variation between 8% and 12%) and KAC (variation between 7% and 9%) savings are achieved across different `PCores` to `ECores` ratios. Fig. 15(b) and (c) compare the performance of GreenMix on the default heterogeneous configuration of 2 `PCores` to 1 `ECores` to the homogeneous configurations of all `ECores` and all `PCores`. For a fair comparison, all three configurations have the same total number of cores. GreenMix achieves an effective balance of QoS, energy consumption, and keep-alive cost on the heterogeneous configuration.

GreenMix provides similar benefits in terms of energy consumption and KAC reduction for different values of information loss parameter ϵ. Recall from Sec. 4.1, the parameter ϵ determines the projection of GreenMix's original optimization problem to a reduced dimensional space. In our experiments, we set $\epsilon = 0.1$, which is a common choice in randomized sketching [6]. This choice provides an energy consumption of 8.64% more than Energy-Opt. and a KAC of 8.79% more than KAC-Opt. By reducing ϵ to 0.05, as expected, GreenMix receives only marginal additional improvement (energy consumption 8.51% more than Energy-Opt. and KAC 8.54% more than KAC-Opt), albeit at a slight increase in overall overhead from 0.31% to 0.43%.

GreenMix dynamically readjusts to changes in – including sudden changes in – function-set composition as a result of its adaptive randomized sketching and probabilistic optimization. For the purposes of challenge-testing GreenMix's effectiveness under large changes, in Fig. 16, we show what happens when the functions in a function set are suddenly varied. Until the 5-hour mark, the functions in a function set vary only at a regular rate of 5-10% (i.e., leave the set due to not being invoked in the near past or enter the set due to new invocations). We observe relatively stable energy consumption and KAC results as GreenMix adapts quickly. Then at the 5-hour mark, the function set is varied by stress-test rates of 100% and 25% (all and 25% the functions in the function set are switched out, and new functions are brought in). Even in the 100%-change scenario, the results become unstable only transiently (remaining within 25% of optimality). In the 25% change scenario, GreenMix stabilizes quickly.

GreenMix's overhead is comparable to practical baselines. GreenMix incurs a 0.31% overhead on average (on `PCore` with warm starts), close to Octopus-Man (0.42%), HyHist (0.27%), and Linux-EAS (0.2%). IceBreaker and Hipster have significantly higher overheads – 5.94% and 10.7%, respectively, due to their FFT and RL-based designs that maintain state for all functions. GreenMix's overhead includes Step-2 (node placement), Step-3 (core selection), and randomized sketching for PE_{Score} and keep-alive time. Sketching overhead is minimal (e.g., 0.127s for a 60s interval) and only incurred at interval boundaries. The stochastic descent variant has a 10.72% overhead. Under double the function load, GreenMix's overhead rises modestly to 0.41%, while stochastic descent, IceBreaker, and Hipster rise to 24.6%, 34.15%, and 21%, respectively. GreenMix's low-dimensional strategy limits overhead growth, unlike IceBreaker, whose FFT cost scales with samples. All results include GreenMix's control-plane overhead on the critical path.

Summary of performance results. Across all three evaluated platforms (ARM big.LITTLE cluster, Intel Alder Lake, 1000-node simulation.), GreenMix achieves energy consumption and keep-alive cost within 10% of optimal solutions, significantly outperforming several state-of-the-art serverless and asymmetric CMP schedulers (which often exceed 40% deviation from optimal). GreenMix's design components, like adaptive randomized sketching and probabilistic optimization, make it scalable and adaptive to dynamic workload changes – for example, when 25% of workload changes, GreenMix maintains performance and stabilizes in less than 5 hours. The only non-trivial algorithmic cost is maintaining the (sketched) function–feature matrices, which we amortize via incremental updates so it remains negligible in practice. GreenMix's overhead is 0.31% of function service time on average, comparable to practical baselines like Linux EAS, but significantly lower than other serverless solutions like IceBreaker.

8 Conclusion

We presented GreenMix, a technique to warm up and schedule serverless functions on asymmetric CMPs to minimize energy consumption and keep-alive cost. GreenMix leverages randomized sketching and probabilistic optimization. We hope that GreenMix will help the community to accelerate the design of new serverless solutions for energy, cost, and performance efficiency.

Acknowledgments

This material is based upon work supported by the Assistant Secretary of Defense for Research and Engineering under United States Air Force Contract No. FA8702-15-D-0001, and United States Air Force Research Laboratory Cooperative Agreement Number FA8750-19-2- 1000. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Assistant Secretary of Defense for Research and Engineering, or the United States Air Force. The United States Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation herein.

References

[1] 2024. Apache OpenWhisk (https://openwhisk.apache.org/). https://openwhisk.apache.org/

[2] 2024. AWS Lambda (https://docs.aws.amazon.com/lambda/). https://docs.aws.amazon.com/lambda/

[3] 2024. Odroid N2+ (https://www.hardkernel.com/shop/odroid-n2-with-4gbyte-ram-2/). https://www.hardkernel.com/shop/odroid-n2-with-4gbyte-ram-2/

[4] Dimitris Achlioptas. 2001. Database-friendly random projections. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 274–281.

[5] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. 2020. Firecracker: Lightweight virtualization for serverless applications. In *17th USENIX symposium on networked systems design and implementation (NSDI 20)*. 419–434.

[6] Nir Ailon and Edo Liberty. 2013. An almost optimal unrestricted fast Johnson-Lindenstrauss transform. *ACM Transactions on Algorithms (TALG)* 9, 3 (2013), 1–12.

[7] Ahsan Ali, Riccardo Pinciroli, Feng Yan, and Evgenia Smirni. 2020. Batch: machine learning inference serving on serverless platforms with adaptive batching. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–15.

[8] Alexandr Andoni, Jiecao Chen, Robert Krauthgamer, Bo Qin, David P Woodruff, and Qin Zhang. 2016. On sketching quadratic forms. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*. 311–319.

[9] Daniel Barcelona-Pons, Marc Sánchez-Artigas, Gerard París, Pierre Sutra, and Pedro García-López. 2019. On the faas track: Building stateful distributed applications with serverless architectures. In *Proceedings of the 20th International Middleware Conference*. 41–54.

[10] Rohan Basu Roy and Devesh Tiwari. 2024. Starship: Mitigating i/o bottlenecks in serverless computing for scientific workflows. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 8, 1 (2024), 1–29.

[11] André Bauer, Haochen Pan, Ryan Chard, Yadu Babuji, Josh Bryan, Devesh Tiwari, Ian Foster, and Kyle Chard. 2024. The globus compute dataset: An open function-as-a-service dataset from the edge to the cloud. *Future Generation Computer Systems* 153 (2024), 558–574.

[12] Luca Becchetti, Marc Bury, Vincent Cohen-Addad, Fabrizio Grandoni, and Chris Schwiegelshohn. 2019. Oblivious dimension reduction for k-means: beyond subspaces and the Johnson-Lindenstrauss lemma. In *Proceedings of the 51st annual ACM SIGACT symposium on theory of computing*. 1039–1050.

[13] Ella Bingham and Heikki Mannila. 2001. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. 245–250.

[14] James Cadden, Thomas Unger, Yara Awad, Han Dong, Orran Krieger, and Jonathan Appavoo. 2020. SEUSS: skip redundant paths to make serverless fast. In *Proceedings of the Fifteenth European Conference on Computer Systems*. 1–15.

[15] Joao Carreira, Pedro Fonseca, Alexey Tumanov, Andrew Zhang, and Randy Katz. 2019. Cirrus: A serverless framework for end-to-end ml workflows. In *Proceedings of the ACM Symposium on Cloud Computing*. 13–24.

[16] Hongsuk Chung, Munsik Kang, and Hyun-Duk Cho. 2012. Heterogeneous multi-processing solution of Exynos 5 Octa with ARM big. LITTLE technology. *Samsung White Paper* (2012).

[17] Marcin Copik, Grzegorz Kwasniewski, Maciej Besta, Michal Podstawski, and Torsten Hoefler. 2021. Sebs: A serverless benchmark suite for function-as-a-service computing. In *Proceedings of the 22nd International Middleware Conference*. 64–78.

[18] George E Dahl, Jack W Stokes, Li Deng, and Dong Yu. 2013. Large-scale malware classification using random projections and neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 3422–3426.

[19] Jesse Donkervliet, Animesh Trivedi, and Alexandru Iosup. 2020. Towards supporting millions of users in modifiable virtual environments by redesigning minecraft-like games as serverless systems. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*.

[20] Dmitriy Fradkin and David Madigan. 2003. Experiments with random projections for machine learning. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. 517–522.

[21] Peter Frankl and Hiroshi Maehara. 1988. The Johnson-Lindenstrauss lemma and the sphericity of some graphs. *Journal of Combinatorial Theory, Series B* 44, 3 (1988), 355–362.

[22] Alexander Fuerst and Prateek Sharma. 2021. FaasCache: keeping serverless computing alive with greedy-dual caching. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 386–400.

[23] Mohammad Ghavamzadeh, Alessandro Lazaric, Odalric Maillard, and Rémi Munos. 2010. LSTD with random projections. *Advances in Neural Information Processing Systems* 23 (2010).

[24] Jashwant Raj Gunasekaran, Prashanth Thinakaran, Nachiappan C Nachiappan, Mahmut Taylan Kandemir, and Chita R Das. 2020. Fifer: Tackling resource underutilization in the serverless era. In *Proceedings of the 21st International Middleware Conference*. 280–295.

[25] Abhinav Gupta, Adithyavairavan Murali, Dhiraj Prakashchand Gandhi, and Lerrel Pinto. 2018. Robot learning in homes: Improving generalization and reducing dataset bias. *Advances in neural information processing systems* 31 (2018).

[26] Udit Gupta, Young Geun Kim, Sylvia Lee, Jordan Tse, Hsien-Hsin S Lee, Gu-Yeon Wei, David Brooks, and Carole-Jean Wu. 2022. Chasing carbon: The elusive environmental footprint of computing. *IEEE Micro* 42, 4 (2022), 37–47.

[27] Jarvis Haupt and Robert Nowak. 2006. Signal reconstruction from noisy random projections. *IEEE Transactions on Information Theory* 52, 9 (2006), 4036–4048.

[28] Joseph M Hellerstein, Jose Faleiro, Joseph E Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu. 2018. Serverless computing: One step forward, two steps back. *arXiv preprint arXiv:1812.03651* (2018).

[29] Scott Hendrickson, Stephen Sturdevant, Tyler Harter, Venkateshwaran Venkataramani, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. 2016. Serverless computation with openlambda. In *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*.

[30] Wendyam Eric Lionel Ilboudo, Taisuke Kobayashi, and Kenji Sugimoto. 2020. Tadam: A robust stochastic gradient optimizer. *arXiv preprint arXiv:2003.00179* (2020).

[31] Laurent Jacques. 2015. A quantized Johnson–Lindenstrauss lemma: The finding of Buffon's needle. *IEEE Transactions on Information Theory* 61, 9 (2015), 5012–5027.

[32] Jananie Jarachanthan, Li Chen, Fei Xu, and Bo Li. 2021. AMPS-Inf: Automatic Model Partitioning for Serverless Inference with Cost Efficiency. In *50th International Conference on Parallel Processing*. 1–12.

[33] Thathachar S Jayram and David P Woodruff. 2013. Optimal bounds for Johnson-Lindenstrauss transforms and streaming problems with subconstant error. *ACM Transactions on Algorithms (TALG)* 9, 3 (2013), 1–17.

[34] Zhipeng Jia and Emmett Witchel. 2021. Nightcore: efficient and scalable serverless computing for latency-sensitive, interactive microservices. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 152–166.

[35] William B Johnson. 1984. Extensions of Lipschitz mappings into a Hilbert space. *Contemp. Math.* 26 (1984), 189–206.

[36] Kostis Kaffes, Neeraja J Yadwadkar, and Christos Kozyrakis. 2019. Centralized core-granular scheduling for serverless functions. In *Proceedings of the ACM Symposium on Cloud Computing*. 158–164.

[37] Kasper Green Larsen and Jelani Nelson. 2017. Optimality of the Johnson-Lindenstrauss lemma. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 633–638.

[38] Hao Li, Xuefei Xu, Jinkui Ren, and Yaozu Dong. 2019. ACRN: a big little hypervisor for IoT development. In *Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*. 31–44.

[39] Jean-Pierre Lozi, Baptiste Lepers, Justin Funston, Fabien Gaud, Vivien Quéma, and Alexandra Fedorova. 2016. The Linux scheduler: a decade of wasted cores. In *Proceedings of the Eleventh European Conference on Computer Systems*. 1–16.

[40] Ashraf Mahgoub, Karthick Shankar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. 2021. SONIC : Application-aware Data Passing for Chained Serverless Applications. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 285–301.

[41] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. 2018. Ray: A distributed framework for emerging AI applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 561–577.

[42] Djob Mvondo, Mathieu Bacou, Kevin Nguetchouang, Lucien Ngale, Stéphane Pouget, Josiane Kouam, Renaud Lachaize, Jinho Hwang, Tim Wood, Daniel Hagimont, et al. 2021. OFC: an opportunistic caching system for FaaS platforms. In *Proceedings of the Sixteenth European Conference on Computer Systems*. 228–244.

[43] Said Nabi, Muhammad Ibrahim, and Jose M Jimenez. 2021. DRALBA: Dynamic and Resource Aware Load Balanced Scheduling Approach for Cloud Computing. *IEEE Access* 9 (2021), 61283–61297.

[44] Rajiv Nishtala, Paul Carpenter, Vinicius Petrucci, and Xavier Martorell. 2017. Hipster: Hybrid task manager for latency-critical cloud workloads. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 409–420.

[45] Edward Oakes, Leon Yang, Dennis Zhou, Kevin Houck, Tyler Harter, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. 2018. SOCK : Rapid task provisioning with serverless-optimized containers. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 57–70.

[46] Vinicius Petrucci, Michael A Laurenzano, John Doherty, Yunqi Zhang, Daniel Mosse, Jason Mars, and Lingjia Tang. 2015. 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, 246–258.

[47] Mert Pilanci and Martin J Wainwright. 2015. Randomized sketches of convex programs with sharp guarantees. *IEEE Transactions on Information Theory* 61, 9 (2015), 5096–5115.

[48] Qifan Pu, Shivaram Venkataraman, and Ion Stoica. 2019. Shuffling, fast and slow: Scalable analytics on serverless infrastructure. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. 193–206.

[49] Garvesh Raskutti and Michael W Mahoney. 2016. A statistical perspective on randomized sketching for ordinary least-squares. *The Journal of Machine Learning Research* 17, 1 (2016), 7508–7538.

[50] Efraim Rotem, Yuli Mandelblat, Vadim Basin, Eli Weissmann, Arik Gihon, Rajshree Chabukswar, Russ Fenger, and Monica Gupta. 2021. Alder Lake Architecture. In *2021 IEEE Hot Chips 33 Symposium (HCS)*. IEEE, 1–23.

[51] Efraim Rotem, Adi Yoaz, Lihu Rappoport, Stephen J Robinson, Julius Yuli Mandelblat, Arik Gihon, Eliezer Weissmann, Rajshree Chabukswar, Vadim Basin, Russell Fenger, et al. 2022. Intel Alder Lake CPU Architectures. *IEEE Micro* 42, 3 (2022), 13–19.

[52] Rohan Basu Roy, Tirthak Patel, Vijay Gadepally, and Devesh Tiwari. 2022. Mashup: making serverless computing useful for HPC workflows via hybrid execution. In *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 46–60.

[53] Rohan Basu Roy, Tirthak Patel, and Devesh Tiwari. 2022. DayDream: executing dynamic scientific workflows on serverless platforms with hot starts. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–18.

[54] Rohan Basu Roy, Tirthak Patel, and Devesh Tiwari. 2022. IceBreaker: warming serverless functions better with heterogeneity. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 753–767.

[55] Mohammad Shahrad, Rodrigo Fonseca, Íñigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. 205–218.

[56] Vaishaal Shankar, Karl Krauth, Kailas Vodrahalli, Qifan Pu, Benjamin Recht, Ion Stoica, Jonathan Ragan-Kelley, Eric Jonas, and Shivaram Venkataraman. 2020. Serverless linear algebra. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 281–295.

[57] Vikram Sreekanti, Harikaran Subbaraj, Chenggang Wu, Joseph E Gonzalez, and Joseph M Hellerstein. 2020. Optimizing prediction serving on low-latency serverless dataflow. *arXiv preprint arXiv:2007.05832* (2020).

[58] Amoghavarsha Suresh and Anshul Gandhi. 2021. ServerMore: Opportunistic Execution of Serverless Functions in the Cloud. In *Proceedings of the ACM Symposium on Cloud Computing*. 570–584.

[59] Zivojin Sustran and Jelica Protic. 2021. Migration in Hardware Transactional Memory on Asymmetric Multiprocessor. *IEEE Access* 9 (2021), 69346–69364.

[60] Karthik Swaminathan and Augusto Vega. 2021. Hardware Specialization: From Cell to Heterogeneous Microprocessors Everywhere. *IEEE Micro* 41, 6 (2021), 112–120.

[61] Ali Tariq, Austin Pahl, Sharat Nimmagadda, Eric Rozner, and Siddharth Lanka. 2020. Sequoia: Enabling quality-of-service in serverless computing. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 311–327.

[62] Jan Treibig, Georg Hager, and Gerhard Wellein. 2010. Likwid: A lightweight performance-oriented tool suite for x86 multicore environments. In *2010 39th International Conference on Parallel Processing Workshops*. IEEE, 207–216.

[63] Ao Wang, Shuai Chang, Huangshi Tian, Hongqi Wang, Haoran Yang, Huiba Li, Rui Du, and Yue Cheng. 2021. FaaSNet: Scalable and Fast Provisioning of Custom Serverless Container Runtimes at Alibaba Cloud Function Compute. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 443–457.

[64] Siqi Wang, Gayathri Ananthanarayanan, Yifan Zeng, Neeraj Goel, Anuj Pathania, and Tulika Mitra. 2019. High-throughput cnn inference on embedded arm big. little multicore processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 10 (2019), 2254–2267.

[65] Fan Yang, Sifan Liu, Edgar Dobriban, and David P Woodruff. 2021. How to reduce dimension with PCA and random projections? *IEEE Transactions on Information Theory* 67, 12 (2021), 8154–8189.

[66] Hanfei Yu, Athirai A Irissappane, Hao Wang, and Wes J Lloyd. [n. d.]. FaaSRank: Learning to Schedule Functions in Serverless Platforms. ([n. d.]).

[67] Tianyi Yu, Qingyuan Liu, Dong Du, Yubin Xia, Binyu Zang, Ziqian Lu, Pingchao Yang, Chenggang Qin, and Haibo Chen. 2020. Characterizing serverless platforms with serverlessbench. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 30–44.

[68] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. 2019. Mark: Exploiting cloud services for cost-effective, slo-aware machine learning inference serving. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. 1049–1062.

[69] Tian Zhang, Dong Xie, Feifei Li, and Ryan Stutsman. 2019. Narrowing the gap between serverless and its state with storage functions. In *Proceedings of the ACM Symposium on Cloud Computing*. 1–12.

[70] Yanqi Zhang, Íñigo Goiri, Gohar Irfan Chaudhry, Rodrigo Fonseca, Sameh Elnikety, Christina Delimitrou, and Ricardo Bianchini. 2021. Faster and Cheaper Serverless Computing on Harvested Resources. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 724–739.

[71] Laiping Zhao, Yanan Yang, Yiming Li, Xian Zhou, and Keqiu Li. 2021. Understanding, predicting and scheduling serverless workloads under partial interference. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15.

[72] Yuhao Zhu and Vijay Janapa Reddi. 2013. High-performance and energy-efficient mobile web browsing on big/little systems. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 13–24.

Appendix: Artifact Description/Artifact Evaluation

Artifact Description (AD)

A Overview of Contributions and Artifacts

A.1 Paper's Main Contributions

C_1 To the best of our knowledge, GreenMix is the first serverless scheduler to leverage asymmetric multi-processors for serverless workload execution, warm up, and placement.

C_2 GreenMix minimizes energy consumption and keep-alive costs for serverless workloads while meeting QoS targets. For every serverless function, it decides the keep-alive time and core type (PCore or ECore) in asymmetric CMPs for function execution. Making effective scheduling decisions must account for the diverse attributes of simultaneously invoked functions, which present varying chances for energy and keep-alive cost reductions while having different QoS targets. Optimizing this diverse pool of functions is challenging due to their varied invocation patterns and execution traits, and factors such as PCore/ECore friendliness. To solve this challenge, GreenMix uses randomized sketching to map a higher-dimensional optimization space into a lower-dimensional one and carries out optimization in the lower-dimensional space to decide a function's keep-alive time and core-type for execution.

C_3 GreenMix is evaluated with a cluster configuration of ARM big.LITTLE processors and Intel Alder Lake platform.

A.2 Computational Artifacts

A_1 https://doi.org/10.5281/zenodo.10967591

Artifact ID	Contributions Supported	Related Paper Elements
A_1	C_1 & C_2 & C_3	Fig.8 Fig.9 Fig.10 Table 2

B Artifact Identification

B.1 Computational Artifact A_1

Relation To Contributions

This artifact contains scripts to schedule serverless functions following the randomized sketching technique introduced in the paper. It executes functions following a production-grade function invocation trace from Microsoft Azure. The performance of the solution can be evaluated under different cluster and node configurations.

Expected Results

GreenMix performs better energy and keep-alive cost reduction compared to all the competing solutions introduced in the paper. It is closest to theoretically optimal solutions to minimize energy and keep-alive costs. Also, GreenMix violates QoS the least compared to other solutions. The performance benefits of GreenMix are observed across almost all serverless functions that are invoked following the open-sourced production-grade function invocation trace from Microsoft Azure cloud.

Expected Reproduction Time (in Minutes)

The estimated time to run the artifact for the entire Microsoft Azure serverless function invocation trace is 52 hours. However, smaller portions of the trace can be completed in lesser time.

Artifact Setup (incl. Inputs)

Hardware. GreenMix is evaluated on a cluster configuration consisting of 15 Odroid N2+ nodes. It is also evaluated on a configuration with Intel Alder Lake 12th generation big.LITTLE processor.

Software. GreenMix requires Python 3.7, and pip3 as dependencies.

Datasets / Inputs. GreenMix executes serverless functions following production grade serverless function invocation trace from Microsoft Azure Cloud (open-sourced at: Data Link)).

Installation and Deployment. Install `numpy`, `pandas`, and `scikit-learn` for the randomized sketching and optimization algorithm in GreenMix. The system processes function invocation traces provided as CSV files containing execution characteristics for both performance and efficiency cores. For detailed execution instructions, refer to the included `README.txt`. The scheduler makes placement decisions based on the configured cluster topology — adjust parameters in `config.py` to match your infrastructure specifications including number of nodes, P-cores/E-cores per node, and memory limits.

Artifact Execution

The execution process begins with the GreenMix orchestrator processing function invocations through the main scheduler. The system employs randomized sketching to project function characteristics into a low-dimensional space via Gaussian projection matrices, optimized using Gaussian Process regression. The node manager handles function placement considering memory constraints and affinities, while the core selector assigns P-cores or E-cores based on QoS requirements. Performance metrics, including energy consumption and keep-alive costs, are continuously updated. They are fed back into periodic optimization cycles.

Artifact Analysis (incl. Outputs)

For each function execution, data regarding which core type is used for execution, keep-alive time, service time, and energy consumption is collected. This information is sent to the manager node.

Artifact Evaluation (AE)

C.1 Computational Artifact A_1

Artifact Setup (incl. Inputs)

Install Python 3.7+ with required dependencies (`numpy`, `pandas`, `scikit-learn`) for the randomized sketching and Gaussian Process optimization components. Configure cluster parameters in `config.py` including node count, P-cores/E-cores per node, memory limits, and QoS thresholds to match your infrastructure. Prepare

function invocation traces in CSV format with execution characteristics for both core types. Execute the scheduler using the command-line interface detailed in `README.txt`, which processes the input traces and performs function invocation.

Artifact Execution

The main program, `greenmix_main.py`, loads function invocation traces and initializes the randomized sketching, node management, and core selection components. For each invocation, it checks warm/cold status, places functions on nodes, selects P-core or E-core based on QoS constraints, and tracks energy consumption. Periodically, the optimization cycle projects active functions to low-dimensional space using Gaussian Process regression to update PE_{Score} and keep-alive parameters. Results are recorded per-invocation with metrics including service time, energy usage, keep-alive time, and SLA violations in a CSV output file.

Artifact Analysis (incl. Outputs)

The achieved keep-alive cost and energy consumption for function execution with GreenMix should closely match the results presented in Figures 8, 9, and 10 of the paper. The QoS/SLA-wise performance of GreenMix should closely match the results presented in Table 2 of the paper.