CLITE: Efficient and QoS-Aware Co-location of Multiple Latency-Critical Jobs for Warehouse Scale Computers

Tirthak Patel
Northeastern University
patel.ti@husky.neu.edu

Devesh Tiwari
Northeastern University
tiwari@northeastern.edu

ABSTRACT

Large-scale data centers run latency-critical jobs with quality-of-service (QoS) requirements, and throughput-oriented background jobs, which need to achieve high performance. Previous works have proposed methods which cannot co-locate multiple latency-critical jobs with multiple backgrounds jobs while: (1) meeting the QoS requirements of all latency-critical jobs, and (2) maximizing the performance of the background jobs. This paper proposes CLITE, a Bayesian Optimization-based, multi-resource partitioning technique which achieves these goals. CLITE is publicly available at `https://github.com/GoodwillComputingLab/CLITE`.

1 Introduction

Background and Motivation. The key to improving data center utilization and operational efficiency is co-locating latency-critical (LC) jobs with throughput-oriented background (BG) jobs [1, 2, 3, 4, 5]. Unfortunately, achieving desirable co-locations are challenging due to strict quality of service (QoS) requirements of LC jobs - violating latency targets can lead to loss of millions of dollars [6, 7, 8, 9]. Traditional efforts have focused on disallowing co-locations altogether [8, 1, 10], or allowing only certain co-locations based on profiling and instrumentation information [1, 10, 8, 11, 12, 13], or dynamically managing interference at runtime by throttling the resources of throughput-oriented BG jobs in order to meet the QoS of the co-located LC job [14, 5, 15, 16, 17, 18]. However, previous research efforts have been largely conservative and hence, have been limited to co-locating *at most one* LC job with one or more BG jobs. But, with rapid emergence of *microservices* and progressive shift of throughput-oriented jobs toward becoming increasingly latency critical [19, 20, 21, 22, 23, 24, 25], there is a need for methods to "safely" (without violating QoSes) co-locate *multiple* LC jobs with multiple BG jobs on the same physical node to bring next generation of significant improvements in data center operation efficiency and cost-effectiveness. Unfortunately, this problem is quite challenging due to high complexity and dimensionality of the solution space (Sec. 2).

Existing Solutions, Limitations, and Challenges. PARTIES, a recent work, is the only work that aims to co-locate multiple QoS-sensitive LC jobs with BG jobs [26]. PARTIES provides a significant improvement over Heracles [5] which was limited to co-locating only one LC job with multiple BG jobs. PARTIES proposes a simple solution toward meeting the latency targets of multiple co-located LC jobs: it tries incrementally increasing/decreasing one resource (e.g., number of cores, memory capacity, etc.) for one LC job at a time, and assesses the observed performance. This process operates until (hopefully) the QoS of all LC jobs are met and after that "leftover" resources are donated to the BG jobs. As shown in their evaluation [26], this approach - as a first-step toward co-locating multiple LC jobs – is practical and can meet QoS targets for multiple LC jobs. While the core merit of PARTIES is its simplicity, it naturally falls short on other desirable characteristics: optimality, robustness, and efficiency.

First, the source of sub-optimality in the PARTIES approach is its limitation to explore only one dimension (resource) at a time and change the amount in small steps for one LC job. In Sec. 2 and Sec. 5, we show that this can lead to inability to co-locate certain sets of LC jobs or achieve only suboptimal partitioning configuration where joint exploration of multiple dimensions simultaneously is necessary - resource requirements of jobs are complex and benefits of increasing one resource allocation in one dimension may not be realized until other dimensions are appropriately resized. Second, PARTIES inherently carries some sources of inefficiencies in utilizing resources because it ignores the performance of throughput-oriented BG jobs. BG jobs are served in the best effort manner - without taking into account the resource sensitivity of BG jobs. In summary, PARTIES is simple and effective, but it does not achieve the full potential to provide QoS guarantees for co-located LC jobs and enable efficient resource utilization by improving performance of BG jobs.

Therefore, the goal of this paper is to achieve the ultimate goal of co-locating multiple LC and BG jobs together by achieving the following two objectives simultaneously: (1) allocate resources to different LC jobs such that it maximizes the number of LC jobs that can be co-located while satisfying their individual QoS targets, (2) allocate resources to BG jobs such that it maximizes the performance of the BG jobs.

An ideal multiple LC and BG co-location scheduling solution should be optimally efficient, practical and robust. However, it is quite challenging to achieve all desirable properties simultaneously. For example, an optimal resource partitioning strategy can be developed in theory by profiling the jobs or user-provided guidance about resource sensitivity of job. But, it will impractical due to high profiling overhead or need for inputs from users. This approach will not be robust to changes in workload characteristics over time and will be inefficient at resource partitioning when more resource knobs are added. An alternative approach is to build performance models for different jobs and accordingly devise resource partitioning schemes which capture their performance sensitivity toward resources. Unfortunately, developing low-overhead yet accurate performance models is challenging, especially considering the dynamically changing characteristics of co-located workloads [27, 28, 8, 29].

Contributions of This Work. This paper introduces CLITE (pronounced as *"see light"*) [1], for *efficiently* co-locating multiple LC jobs with throughput-oriented BG jobs to meet multiple objectives: satisfy QoS target for all LC jobs and maximize performance of all BG jobs. CLITE takes an unorthodox approach - build simple and less-than-fully-accurate models but uses these "imperfect" models to achieve near-optimal efficiency. This approach makes CLITE low-overhead, practical, and feasible (Sec. 3). CLITE leverages Bayesian Optimization to provide theoretically-grounded resource partitioning of multiple resources (e.g., cores, caches, memory bandwidth, memory capacity, disk bandwidth etc.) among multiple co-located jobs. CLITE provides near-optimal solution without building complex and high-overhead performance models which need profiling or recompilation.

CLITE uses Bayesian Optimization (BO) methods to build low-cost performance model of different resource partitioning configurations by sampling a small number of points in the large configuration space and then, navigates this search space intelligently to find near-optimal configurations. Unfortunately, applying BO to shared resource partitioning is quite challenging and naïve application of BO can lead to worse than simple heuristic results. CLITE proposes new techniques to overcome these challenges and find effective balance between multiple design trade-offs (Sec. 4). CLITE takes a principled approach to explore multiple resource dimensions and jobs simultaneously to (1) effectively co-locate multiple LC and BG jobs, and (2) extract hidden opportunities for improving efficiency by exploiting differences among jobs toward resource sensitivity (Sec. 2).

CLITE's evaluation demonstrates its effectiveness, robustness, and practical feasibility across a range of scenarios and workloads. CLITE's LC job performance is within 5% of the oracle scheme and better than the previously proposed solutions such as PARTIES [26] and Heracles [5], by more than 15% in many cases. CLITE can co-locate a set of resource-hungry and latency-critical jobs while meeting their QoS targets, and can still provide high performance to background jobs, in comparison the competing techniques (including PARTIES and genetic algorithm approach). CLITE is publicly available at `https://github.com/GoodwillComputingLab/CLITE`.

2 Challenges and Opportunities

Contemporary chip multi-processor servers consist of multiple shared resources and several tools exist to partition these shared resources among co-located jobs (Table 1 provides summary of example shared resources and corresponding partitioning methods). Each job can be allocated some fraction of the shared resource using existing isolation tools (e.g., Intel MBA utility can be used to provide 40% and 60% memory bandwidth to two co-located jobs). Unfortunately, determining optimal allocation of these shared resources is quite challenging for several reasons.

[1]CLITE stands for <u>c</u>o-locating <u>l</u>atency-cr<u>i</u>tical and <u>t</u>hroughput-oriented jobs <u>e</u>fficiently.

Table 1: Summary of different shared resources on a CMP server, and their isolation tool/interface. Each shared resource can have multiple *units* that can be shared at different granularities (e.g., LLC sets can have 11 ways and can be shared at single-way granularity).

Shared Resource	Allocation Method	Isolation Tool
CPU Cores	Core Affinity	`taskset`
Last Level Cache (LLC)	Way Partitioning	Intel CAT [30]
Memory Bandwidth	Bandwidth Limiting	Intel MBA [31]
Memory Capacity	Capacity Division	Linux's `memory cgroups`
Disk Bandwidth	I/O Bandwidth Limiting	Linux's `blkio cgroups`
Network Bandwidth	Network B/w Limiting	Linux's `qdisc`

Finding optimal resource partitioning requires exploring a prohibitively large, multi-dimensional search space. Co-locating one latency-critical job with throughput-oriented jobs while meeting a QoS target is already hard even when the configuration space is small and multiple configurations may satisfy the objective [12, 11]. A configuration refers to the resource partition setting for the given set of job (e.g., 1 core and 7 cache set ways to the LC job and 3 cores and 4 cache set ways to the BG job). Co-locating multiple LC and BG jobs, each with their own goals explodes the size and dimensionality of the search space.

Suppose there are N_{res} resources, N_{jobs} co-located jobs, and the r^{th} resource has $N_{\text{units}}(r)$ units of the resource, then the number of total configurations can be expressed as $N_{\text{conf}} = \prod_{r=1}^{N_{\text{res}}} \binom{N_{\text{units}}(r)-1}{N_{\text{jobs}}-1}$ in a $N_{\text{res}} \times N_{\text{jobs}}$-dimensional space. For example, if four co-located jobs share just three resources (e.g., number of cores, memory bandwidth, and memory capacity), each resource with 10 units, then the total number of possible configurations is $592,704$ and the search space has 12 dimensions. Exploring the full search space to find an optimal resource partitioning configuration that satisfies QoS targets and maximizes BG performance is not feasible as it would take several hours. This configuration space can be potentially pruned by collecting and utilizing prior information about job characteristics (e.g., via profiling), but *a desirable solution should not make assumptions about resource consumption characteristics of co-located jobs, require prior information or employ intrusive methods (e.g., profiling, recompilation, instrumentation).*

Developing accurate analytical or empirical performance models to traverse large configuration search space efficiently is challenging, and not adaptable to changes. A promising solution to traversing large search space is developing job performance models that can guide the search process toward promising areas by exploiting the model-learned relationship between the job performance and assigned resource shares (e.g., cache set ways, memory bandwidth). However, developing accurate and practical performance models is quite challenging due to complex interaction among multiple performance-impacting factors [28, 8, 32, 33]. Moreover, such models can be rendered ineffective as the jobs are updated or improved, and the corresponding models need to be rebuilt (i.e., not adaptable).

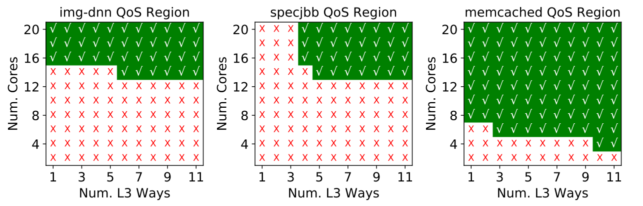


Figure 1: QoS-safe regions for three LC jobs [✓]: multiple configurations can meet QoS, but share of one resource depends on the share of other.

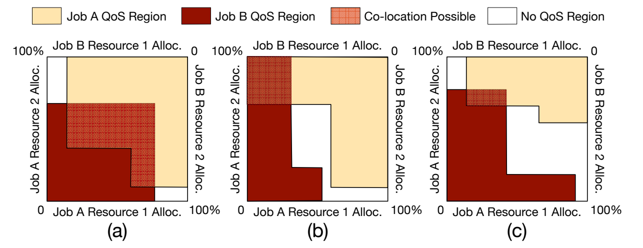


Figure 2: Coordinate descent methods may not always find the optimal configuration (e.g., case (c)), even in a 2-D small search space. Overlapped regions indicate the resource configurations that will meet QoS for both jobs.

Optimizing resource partitioning via coordinate descent (i.e., finding solutions with respect to one dimension/resource at a time) is sub-optimal and prone to instability in a dynamic environment. A desirable solution should explore multiple resource configurations simultaneously to obtain more efficient resource configurations. Real-world jobs exhibit varying performance sensitivities toward different resources, and the degree of sensitivity depends on the share of other resources. To build an intuition, we provide a simple example using only two shared resources (Fig. 1) that shows the "QoS-safe regions" for three representative LC jobs used in this study (details of workloads and experimental platform in Sec. 5). We observe that img-dnn job can achieve its QoS with multiple configurations and share of one resource depends on the share of other resource (e.g, 16 cores with 1 way or 14 cores but 6 ways) - we refer to this property as "resource equivalence class" - where two resource configurations are interchangeable for QoS requirements although they may yield different performances. QoS-safe regions for specjbb and memcached are significantly different, but they also exhibit resource equivalence class property. This property is key to achieving QoS when co-locating multiple LC jobs with diverse resource sensitivity. But, methods that explore only one dimension at a time while keeping the allocations in the other directions fixed are bound to fail to exploit this opportunity. PARTIES also implements a simpler variant of this approach [26]. While simple to implement, such an exploration is not specifically guided toward global optimum and can lead to sub-optimal results, poor performance for non-smooth multi-variable objective functions, and may be unable to stabilize when the co-locations change quickly [34, 35, 36]. Fig. 2 provides a simple example using only two resources and two jobs with simple QoS curves that demonstrates the inefficiency of this approach.

In Fig. 2(a), coordinate descent can easily achieve a resource configuration that satisfies the QoS of both jobs (equal division of resources as the starting point). However, as shown in Fig. 2(b), the success of coordinate descent depends on the initial point. In this case, exploring configurations in small steps around the equal division will not work, but one might find optimal configuration by starting at left-top corner (100% and 0%) and incrementing in steps. Unfortunately, as Fig. 2(c) shows, finding an optimal configuration with exploring only one dimension at a time may not provide a solution until a suitable starting point is chosen - since no apriori information (e.g., QoS-safe region curves) is available beforehand, one cannot easily choose exploration points efficiently. As more dimensions and applications are added, finding optimal solutions in a multi-dimensional large search space gets extremely challenging and infeasible if naïve methods, such as exploring one dimension at a time while keeping the allocations in the other directions fixed, are employed. CLITE leverages Bayesian Optimization to overcome this challenge and explore multiple resource dimensions simultaneously by exploiting the resource equivalence class property.

3 CLITE: Overview

CLITE adopts the following design principle: "solving complex problems with near-optimality using theoretically-grounded yet practically viable approach". In that spirit, CLITE neither requires accurate and complex performance models, nor does it need apriori information about different co-located jobs to estimate optimal resource partition. Instead, CLITE uses *just-accurate-enough* models to estimate near-optimal resource partitioning configuration to meet its objectives. Using just-accurate-enough models allows CLITE to lower the overhead associated with such models and improve its feasibility, while providing an effective solution (Sec. 5.2 shows that CLITE achieves near-optimal results).

CLITE needs to partition multiple resources (e.g., number of compute cores, shared cache space, memory bandwidth, memory capacity and disk bandwidth) among competing jobs, but exhaustively exploring different partitioning configurations and observing their respective performance to find the optimal configuration is prohibitively expensive. CLITE uses BO to intelligently navigate through this exploration to find efficient resource partitioning configurations. Next, we discuss how BO works and how CLITE leverages BO to meet its objectives effectively.

3.1 How does Bayesian Optimization work?

BO is a generic, theoretically-grounded, black-box method for solving an optimization problem of maximizing an unknown objective function, f. BO does not need to know the relationship between the input and the objective function, but the objective function can be queried multiple times with different inputs. Traditionally, the aim of the optimization is to figure out the input x^* that results in the most optimal value of the objective function $f(x)$ with *minimum possible queries*. Mathematically, BO can be expressed as following (where $\mathscr{X}$ is the search space of interest):

$$x^* = \underset{x \in \mathscr{X}}{\operatorname{argmax}} f(x) \qquad (1)$$

In the context of CLITE, the input, x, to the objective function, f, is essentially a *configuration* which allocates different units of each resource to different co-located jobs (an example in the context of CLITE: 3 cache set ways to LC job 1, 4

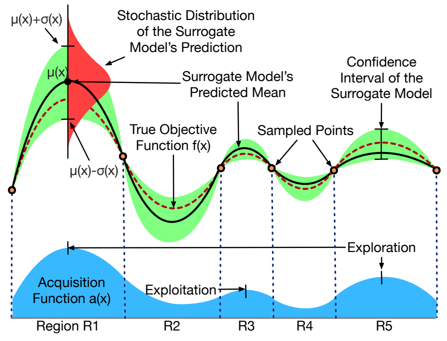

μ(x)+σ(x)
Stochastic Distribution of the Surrogate Model's Prediction
μ(x)
Surrogate Model's Predicted Mean
Confidence Interval of the Surrogate Model
True Objective Function f(x)
Sampled Points
μ(x)-σ(x)
Exploration
Acquisition Function a(x)
Exploitation
Region R1
R2
R3
R4
R5

Figure 3: Surrogate model attempts to model the unknown objective function and acquisition function helps navigate the search space intelligently.

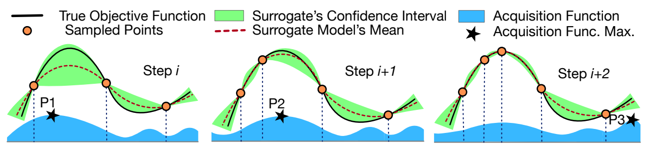


Figure 4: The acquisition function intelligently "explores" and "exploits" the space and the surrogate model iteratively becomes more accurate.

cache set ways to LC job 2, 30% memory bandwidth to LC job 1, and 40% memory bandwidth to LC job 2, and the rest of the resources to the BG job). All such possible *resource partition configurations* comprise the search space $\mathscr{X}$ and are referred as *sample points* in the space. The objective function, f, is unknown because CLITE does not know the relationship between the input configuration and the objective function (i.e., the statistical function that expresses the relationship between a given resource partitioning configuration and corresponding outcome in terms of latency targets for LC jobs and performance of BG jobs is not known and may not exist in a closed form). However, CLITE can evaluate the value of the objective function, f, given an input x (resource partition configuration) by running the system with given configuration and observing the performance of all co-located jobs. The value of the objective function indicates the performance-level of BG jobs and if the LC jobs have met their QoS targets. However, unlike traditional cases where the optimization problem is to simply maximize the objective function [37, 38, 39, 40, 41, 42], CLITE requires careful construction of the objective function. This challenging aspect of CLITE and corresponding solution is discussed later in Sec. 3.3. CLITE uses BO to find the optimal value of the objective function with minimum number of function evaluations (i.e., running minimum number of configurations).

Algorithm 1 CLITE Bayesian Optimization.

```
1: Input: Initial resource configurations ($S_{init}$), termination condition ($T$), & maximum iterations ($N_{\text{iter}}$).
2: for $i = 1$ to $N_{\text{iter}}$ do
3:     Update the surrogate model $\mathscr{M}$.
4:     Compute the acquisition function $a(x)$.
5:     Find the next sample to evaluate.
6:     Run the system with the selected configuration $x$.
7:     Observe performance & add it to the existing set.
8:     if $T$ is met then break
9: Output: Best resource partition configuration $x^*$.
```

To identify the global optimum of the objective function, BO intelligently explores the search space by evaluating the objective function with different input samples (configurations). As it explores the sample space, it builds a stochastic model of the unknown objective function and keeps updating it, and uses this knowledge to identify which samples to evaluate next such that it reaches near the global optimum. The two key components that enable this are: (1) the *surrogate model* (also, known as prior or belief model), and (2) the *acquisition function*. BO uses a surrogate model, $\mathscr{M}$, to stochastically estimate the performance of different points (configurations) in space. This stochastic model is improved iteratively during the space exploration process as more and more points are sampled. Fig. 3 shows a snapshot of the state of a BO execution during exploration where six points are already sampled and BO has estimates for the rest of the points as predicted by the surrogate model. We note that the surrogate model predicts a range of values for non-sampled points - it consists of both a predicted mean ($\mu(x)$) and variance ($\sigma(x)$). The variance captures the model's "uncertainty" at a given point (e.g., x in Fig. 3). BO is seeded a small initial set of points with corresponding values of the objective function.

Next, BO needs to steer itself toward global optima by sampling the "most promising points". BO uses the acquisition function to move in the right direction during the space exploration process i.e., an acquisition function $a(x)$ dictates which points to sample next such that BO moves closer to the most-performant samples in the space while sampling only a small number of points. The role of the acquisition function is to act as a "cheap proxy" for the unknown objective function such that it returns high values for the most promising non-sampled points (Fig. 3). As shown in Fig. 3, acquisition function returns different values for different non-sampled points at a given stage; essentially the acquisition function helps BO explore different neighborhoods. It strikes a balance between "exploration" (exploring the areas of the configuration space with large uncertainty) and "exploitation" (exploiting the vicinity of neighborhood where the predicted mean value of the surrogate model is high). In Fig. 3, the acquisition function suggests BO to "explore" the leftmost region (R1) next. Note that as more points are sampled, the surrogate model is updated and the acquisition function is reevaluated after each sample. To illustrate how BO works iteratively, we provide a simple example in Fig. 4. Fig. 4 shows the process as the number of samples grow from 3 to 5. The acquisition function selects a region where the uncertainty is highest, and then, exploits this region by sampling two points, $P1$ and $P2$, in this region (step i and $i+1$) before moving to "exploring" a new neighborhood at step $i+2$ ($P3$). These steps, as expected, decrease the uncertainty of the model in the second leftmost region, and result in a new optimum. This iterative process is summarized in Algorithm 1.

3.2 Why does CLITE use Bayesian Optimization?

CLITE builds a BO based technique to find an efficient resource partitioning configuration among competing jobs for several reasons. First, BO neither needs to know or assume anything about the objective function, nor put restriction on the underlying objective function (i.e., it is non-parametric). Second, BO does not need to know the derivative of the objective function to guide its exploration process for the optimal solution. Third, BO does not require to know the complex interactions among the parameters of the objective function (e.g., increasing the cache set ways in the shared cache for a job has a visible effect on its latency only after the memory bandwidth is increased beyond a certain share), but instead it captures these interactions and relationships automatically during execution. Thus, BO can effectively adjust multiple resources simultaneously, unlike gradient descent methods which are limited to exploring one dimension at a time.

Alternative black-box optimization methods include deep neural networks, simulated annealing, reinforcement learning, and genetic algorithms. The main advantage of BO over these alternative methods is its ability to find highly-performant solution with limited number of sample evaluations, this is especially beneficial when the evaluation of the objective function is costly and the optimized solution has to be obtained quickly - as is the case with CLITE. Further, alternative methods such as deep neural networks and reinforcement learning require large amount of training data for accurate predictions. These methods are not suitable for handling uncertainty and noise, and require expensive space search before reaching efficient solution configurations. Our evaluation confirms that BO's continuous updates to the surrogate model and acquisition function enables it to outperform genetic algorithm inspired techniques even with a smaller number of samples (Sec. 5).

3.3 Challenges in Designing Efficient BO for CLITE

While promising in theory, developing a practical and efficient BO-based strategy is quite challenging in CLITE's problem space for several reasons. First, BO is well-known to have limited effectiveness in high dimensional parameter space - unfortunately, CLITE needs to operate in a high dimensional search space since it deals with large number of possible resource partition configurations (Sec. 2). Second, BO needs to have a suitable objective function that guides the search space exploration. This objective function assigns a value of objective for a given sample (resource partition configuration) to assess its closeness to the optimal value of the unknown objective function. For traditional scenario, this is a scalar value that BO attempts to maximize (or minimize) and is smooth in nature. Unfortunately, the objective function for CLITE is a set of goals (meeting QoS targets) and maximization of the performance of the background jobs. CLITE's objective needs to be carefully formulated to ensure that it is reasonably smooth. A non-smooth function can render the search exploration process ineffective and may not steer the exploration toward promising regions. Third, BO's effectiveness is influenced heavily by the initial configurations that it samples at first, called bootstrapping configurations. Instead of random sampling used traditionally, CLITE needs to carefully construct bootstrapping configurations to accelerate the search process and lower the overhead. Next, we discuss the design choices and trade-offs that CLITE makes for an efficient Bo-based resource partitioning solution.

4 CLITE: Design and Implementation

Surrogate Model. CLITE chooses Gaussian Process (GP) as the surrogate model [43]. Gaussian Process is a distribution over an objective function specified by a mean and a covariance. Essentially, possible evaluation results of the objective function for a non-sampled resource configuration are assumed to follow a multivariate Gaussian distribution.

CLITE chooses GP as the surrogate model due to multiple desirable properties. GP does not make assumptions about the objective function or characteristics of the co-located jobs and hence, provides the flexibility to be trained to model the underlying hidden objective function accurately enough. Compared to other alternatives such as random forests, GP provides better extrapolation quality for non-sampled configuration points away from the neighborhood of the sampled configurations [44]. A well-known drawback of using GP is its associated computational overhead for the regression inference. CLITE mitigates this overhead by carefully limiting the number of sampled data points (discussed below) instead of employing sparsification and approximation techniques which result in poor uncertainty estimates and high degree of oscillations in predictions [44]. CLITE uses the Matérn covariance kernel with the GP model. The covariance determines the similarity between two resource partition configurations. Matérn covariance kernel is suitable for CLITE since it does not require restrictions on strong smoothness [43, 45] - a desirable property for CLITE as discussed later.

Acquisition Function. The design trade-offs in selecting an acquisition function are (1) evaluating the acquisition function should be relatively inexpensive, and (2) to find the global optimum, the acquisition function should balance exploration and exploitation (Sec. 3.1 and Fig. 3).

If evaluating the acquisition function is as expensive as the objective function, f, then, the purpose of employing an acquisition function is defeated since it does not *quickly* guide the BO toward promising samples. However, cheap acquisition functions such as probability of improvement (PI) suffer from inability to find the balance between exploitation and exploration; it often gets stuck in local optima due to limited exploration capabilities. More recent sophisticated acquisition functions such as predictive entropy search and upper confidence bound based methods provide closer to ideal balance between exploration and exploitation, but incur high computational cost - which is suitable for offline decision making problems, but not suitable for CLITE where the sampling process and corresponding decision making process are online and time-constrained. CLITE chooses the Expected Improvement (EI) method that provides practical balance between exploration vs. exploitation at a low evaluation cost.

EI has a desirable property of fast convergence in practice and has been shown to be effective for complex real jobs [43]. To improve the exploration vs. exploitation trade-offs, CLITE augments traditional EI method with a factor ζ that is shown to be effective for such a purpose by Lizotte et al. [46], low values of ζ such as 0.01 work well in practice. Mathematically, The EI of a vector x is calculated using Eq. 2. $\mu(x)$ is the estimated mean at x, $\hat{x}$ is the current optimum, ζ is a parameter which dictates the exploration vs. exploitation aspect of the BO, $z = \frac{(\mu(x)-\hat{x}-\zeta)}{\sigma(x)}$, $\Omega(z)$ & $\omega(z)$ are standard normal CDF and PDF of z, and $\sigma(x)$ is the standard deviation of x. The x with maximum $E(x)$ value is used as next sample.

$$E(x) = \begin{cases} (\mu(x)-\hat{x}-\zeta)\Omega(z)+\sigma(x)\omega(z) & \text{if } \sigma(x) > 0 \\ 0 & \text{if } \sigma(x) = 0 \end{cases} \quad (2)$$

Assigning Scores to the Objective Function Evaluation. In a traditional BO implementation, evaluating the objective function returns a single value (e.g., throughput of the system, avg. waiting time of the queue) to be maximized or minimized. Unfortunately, CLITE cannot apply traditional BO since CLITE needs to satisfy multiple criteria (QoS of LC jobs and maximize certain goals (performance of BG jobs). But, to make decisions about which configurations to evaluate next, CLITE still needs to assess the "goodness" of a configuration when evaluated (i.e., the system is run for a short duration under the given resource partition configuration).

To this end, CLITE designs a *score function* that assigns scores to objection function evaluation (i.e., a objective score is assigned at the end of the period when the system is run under the given resource partition configuration). This score function guides CLITE to search in the right direction in the large configuration space. To achieve that effectively, the score function needs to be as smooth as possible. For example, consider a scenario where only a few hundred configurations out of thousands of configurations satisfy the QoSes of all latency-sensitive jobs. If the score function returns 0 for all configurations which do not meet the QoS requirements, the score of the entire search space will be flat with sudden jumps at a few hundred samples. Thus, this score function will not help BO steer in the right direction as most of the configurations sampled will have the same objective value of 0, providing the BO with no specific direction.

We construct a normalized objective function which returns values between 0 (worst case scenario, no LC job meets its QoS) and 1 (ideal scenario, all LC jobs meet QoS and BG jobs achieve the same performance as if they were running in isolation). CLITE's first objective is to meet the QoS targets of all LC jobs before optimizing for BG jobs. To capture this intent, CLITE's score function can not attain a value higher than 0.5 if the QoS target for all the LC jobs are not met, irrespective of the BG jobs' performances. Only when the the QoS target for all the LC jobs are met, the performances of BG jobs are taken into account. Hence the score function has two modes, as described by Eq. 3. For configurations which do not meet the QoS requirements of all co-located LC jobs, the score function returns a value according to the first mode: N_{LC} is the number of LC jobs, QoS-Target$_n$ is the QoS of n^{th} LC job, and Current-Latency$_n$ is the latency of the LC job under current resource configuration. Note that this mode would return a value of 0.5 at maximum.

$$\text{Score} = \begin{cases} \frac{1}{2} \times \sqrt[N_{\text{LC}}]{\prod_{n=1}^{N_{\text{LC}}} \min(1.0, \frac{\text{QoS-Target}_n}{\text{Current-Latency}_n})} & \text{if } \exists \text{ LC, QoS not met} \\ \frac{1}{2} + \frac{1}{2} \times \sqrt[N_{BG}]{\prod_{n=1}^{N_{BG}} \frac{\text{Colo-Perf}_n}{\text{Iso-Perf}_n}} & \text{otherwise} \end{cases} \quad (3)$$

For configurations, which meet the QoS requirements of all LC jobs, the second mode of the score function takes effect, where N_{BG} is the number of batch jobs, Colo-Perf$_n$ is the performance of the n^{th} batch job under current co-located resource configuration, and Iso-Perf$_n$ is its baseline performance under maximum resource allocation (performance under isolation, as sampled during the initialization phase). This indicates to CLITE that meeting QoS can only improve the objective to half of its maximum value, and that it needs to keep searching for better configurations which not only meet the QoS requirements, but also maximize the performance of batch jobs (also analyzed in Sec. 5.2). If no batch jobs are co-located, then CLITE maximizes the performance of all latency-sensitive jobs past meeting their QoSes - to improve the overall efficiency of the system. N_{BG} is simply replaced by N_{LC} in this scenario in Eq. 3. The goal of CLITE is to find a resource allocation which maximizes this score.

Selecting Bootstrapping Configuration Samples. Carefully selecting the initial sample points is critical for a BO-based method as it helps BO guide toward the near-optimal solution quickly. In traditional BO-based optimization problems, random sampling may work well, esp. if the "good" samples cannot be easily identified beforehand and the optimization process does not have tight time constraints [47, 46]. However, it is more desirable for CLITE to accelerate the search process to make optimal decisions quickly. To this end, CLITE carefully constructs the bootstrapping set of resource partitioning configuration to help BO converge faster. This set includes two types of samples: (1) each resource is divided as equally as possible among all co-located jobs (e.g., in case of four co-located jobs, each one gets 25% of the memory bandwidth, 3 set ways for all jobs except one in a 11-way set associate cache) - the purpose of these samples is to provide BO with a good starting point, (2) each job gets the maximum possible allocation across all resources, and the remaining co-located jobs get 1 unit of each resource. These configurations are the extrema of the search space and serve as a useful informed guide for BO. Moreover, sampling these initially helps CLITE quickly identify jobs which will not meet their QoS even under maximum allocation given a set of co-located jobs. These jobs can be immediately scheduled elsewhere without wasting any BO cycles. Together, such configurations better inform BO about the underlying unknown objective function and accelerate the search process.

Mitigating High Dimensionality Limitations. As discussed earlier, CLITE needs to operate and be effective in a high-dimensional search space, but scalability and effectiveness of BO methods begin to struggle as the search space grows due to increase in the evaluation cost of acquisition function [47, 37, 46, 45]. Prior works have focused on addressing this limitation by focusing on only a limited set of "active" dimensions, projecting performance on untested dimensions, or decomposing the objective function [37, 47]. Unfortunately, these methods cannot be applied to CLITE since all dimensions are active and need to be tuned. Nonetheless, CLITE leverages recent advances in BO literature that provide practical effectiveness using the concept of "dropout-copy" [47] - where randomly selected dimensions are replaced by their best value sampled so far. CLITE employs dropout-copy to hold some parameters to a constant (i.e., resource allocation for certain jobs), while others are tuned.

CLITE exploits this concept further to make it yield more benefits in its specific scenario: instead of selecting random dimensions for dropout, it looks at already sampled configurations and picks the job which is performing the best so far, i.e., has met or is closest to meeting its QoS. This job's resource allocation is held constant to the allocation with which it performed the best, while the rest of the jobs' resource

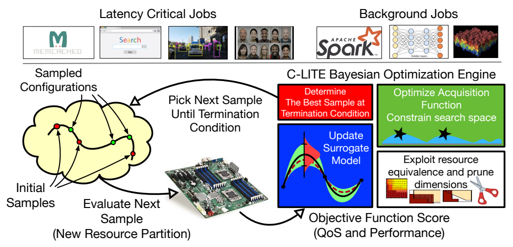

Latency Critical Jobs
Background Jobs
Sampled Configurations
C-LITE Bayesian Optimization Engine
Pick Next Sample Until Termination Condition
Determine The Best Sample at Termination Condition
Optimize Acquisition Function
Constrain search space
Update Surrogate Model
Exploit resource equivalence and prune dimensions
Initial Samples
Evaluate Next Sample (New Resource Partition)
Objective Function Score (QoS and Performance)

Figure 5: CLITE: Putting it all together.

allocations are searched over to find the one with highest expected improvement. We choose only one job for dropout as it has been shown that dropping out too many dimensions can cause the algorithm to not find the optimal value [47]. Beside improving efficiency, CLITE's dropout-copy mechanism also helps it focus more on jobs which are not performing well, which helps CLITE converge faster.

To further address the challenge of multi-dimensional configuration space and to accelerate the search process, CLITE employs constrained execution which prunes "likely-to-be-sub-optimal" resource partition configurations. It does so by applying individual constraints on each resource for each job and solving a constrained optimization problem, thus constraining the input space when optimizing the acquisition function $a(x(j,r))$ in Eq. 4, where $x(j,r)$ is the allocation of resource r to job j. CLITE uses constrained Sequential Least Squares Programming (SLSQP) to achieve efficient and accurate optimization for the following problem formulation. The first constraint (Eq. 5) is that the minimum allocation per resource is 1 unit and maximum allocation is the number of remaining units after every other job has received at least 1 unit. In Eq. 5, J is the set of co-located jobs, R is the set of resources, $N_{\text{units}}(r)$ is the number of units of resource r, and $N_{\text{jobs}} = |J|$ is the number of co-located jobs. The second constraint is that all allocations must sum up to the maximum number of available units per resource (Eq. 6).

$$\text{maximize } a(x(j,r)) \tag{4}$$

$$\text{s.t. } \forall j,r \in J,R \rightarrow 1 \leq x(j,r) \leq N_{\text{units}}(r) - N_{\text{jobs}} + 1 \tag{5}$$

$$\& \ \forall r \in R \rightarrow \sum_{j \in J} x(j,r) = N_{\text{units}}(r) \tag{6}$$

Choice of Termination Condition. CLITE is agnostic to the number of resources, number of jobs, and job characteristics for better scalability and portability of the approach. Hence, using traditional static termination condition [44, 48, 49] would result in sub-optimal results as the problem size and characteristics vary across the set of co-located jobs. To address this issue, CLITE employs a termination condition based on the acquisition function which is inherently aware of the co-located job characteristics. When the "expected improvement" drops below a certain threshold, CLITE terminates its search process. This threshold can be as low as 1%, but needs to be scaled according to the number of co-located jobs to take into the account that the curve of drop in the expected improvement is slower as the number of co-located jobs increase. Our evaluation results show that tunable parameter provides CLITE the ability to manage the trade-offs between quality of results and convergence time (Sec. 5).

CLITE: Putting it all together. Overall, the decision making flow and implementation of CLITE are shown in Fig. 5. CLITE partitions shared resources (e.g., cores, caches, memory, disk, etc.) dynamically at fine granularity without user/job input, or recompilation.

CLITE runs as a single-thread BG job in order to avoid interference with LC jobs. It is first seeded with a careful set of initial configurations to guide CLITE's surrogate model in the right direction and disallow infeasible co-locations. Then, CLITE uses its acquisition function to decide which configuration setting to evaluate next. CLITE applies previously discussed optimizations to reduce the dimensionality of the sample space and constrain it. CLITE observes the performance of each co-located job using performance counters and checks whether LC jobs have met their QoS targets. The observation period is configured to be two seconds currently to ensure that CLITE has sufficient number of queries to calculate QoS violations with high statistical significance. Strict QoS targets (e.g., 95%-ile) require enough samples to be collected to accurately estimate the current QoS. Also, if the observation period is too small, two consecutive configuration samples may interfere to an observable degree (e.g., cache data evicted too quickly from shared L3 cache, context switch misses at the L1 and L2 cache, etc.). Therefore, CLITE has configured the observation period to be two seconds, and it has flexibility to be configured as needed. Next, guided by the drop rate in the expected improvement curve, CLITE's BO engine operates until a near-optimal configuration is found and implemented. Thereafter, performance for all jobs is periodically monitored. If the observed performance or the job mix changes, CLITE can be reinvoked to determine new optimal resource partition for co-located jobs.

5 Evaluation and Analysis

5.1 Experimental Methodology

Table 2: Experimental testbed configuration.

Component	Specification
CPU Model	Intel(R) Xeon(R) Silver 4114
Number of Sockets	1
Processor Speed	2.20GHz
Logical Processor Cores	20 Cores (10 physical cores)
Private L1 & L2 Cache Size	32KB and 1024KB
Shared L3 Cache Size	14080 KB (11-way set associative)
Memory Capacity	46 GB
Operating System	Ubuntu 18.04.1 LTS (4.15.0-36-generic)
SSD Capacity	500 GB
HDD Capacity	2 TB

Table 2 summarizes our testbed platform used for CLITE evaluation. We use the resource partitioning tools (e.g., Intel CAT, MBA) listed in Table 1 to provide resource allocation among competing jobs. Table 3 lists all the LC and BG workloads used for CLITE evaluation. LC workloads are obtained from the Tailbench benchmark suite [50] specifically developed to capture characteristics of LC workloads. The BG workloads are taken from the PARSEC benchmark suite [51] - a widely used benchmark suite for evaluating throughput-oriented workloads. To determine the QoS requirements of the LC workloads, we run them in isolation at different loads or queries-per-second (QPS). This step is required for estimating the QoS tail latency for different workloads and

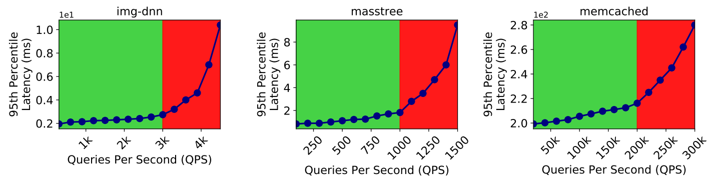


Figure 6: The 95th %-ile QoS tail-latency of the LC jobs is taken from the knee of the queries-per-second (QPS) vs. tail-latency curves. The corresponding QPS is considered as the maximum load for that job.

is necessary for evaluating the success of any co-location method (i.e., it is not specific to the co-location method being evaluated). We note their 95th percentile tail-latencies at these different loads, as shown in Fig. 6 - similar to methodology used by other co-location papers [26, 14, 52]. The QoS tail-latency of the LC workloads is the knee of these curves and the corresponding QPS is the maximum load for that workload. Note, some loads (<20%) can have QPS as low as 100 (e.g., masstree) and may require one or two seconds to calculate the 95th percentile. Finally, we do not directly control the CPU power using `cpufreq` or DVFS because we observed the CPU frequency does not have a large operating range if multiple cores are active, as is the case when multiple jobs are co-located and most cores are being utilized.

Table 3: LC and BG workloads driving CLITE evaluation.

Latency-Critical (LC) Workloads	
img-dnn	Image recognition [50]
masstree	Key-value store [50]
memcached	Key-value store [53] with Mutilate [54] load generator
specjbb	Java middleware [50]
xapian	Online search (inputs: English Wikipedia [50]
Background (BG) Workloads	
blackscholes (BS)	Option pricing with Black-Scholes Partial Diff. Equation (PDE) [51]
canneal (CN)	Simulated cache-aware annealing to optimize chip design [51]
fluidanimate (FA)	Fluid dynamics for animation with Smoothed [51]
freqmine (FM)	Frequent itemset mining [51]
streamcluster (SC)	Online clustering of an input stream [51]
swaptions (SW)	Pricing of a portfolio of swaptions [51]

Competing Co-location Scheduling Policies. We compare CLITE with the following competing policies:
Random-Plus Search (RAND+). RAND+ stochastically selects a configuration to sample from a set of all possible configurations using a uniform distribution. To avoid sampling similar configuration multiple times, it selectively discards a new sample if the Euclidean distance between the selected configuration and existing ones are smaller than a threshold.
Genetic-Algorithm Inspired Search (GENETIC). GENETIC starts by sampling multiple configurations. It selects the two with the highest objective function values and generates new configurations by combining the resource allocations of the two configurations in different forms ("crossover"). Then, the generated combinations are tweaked using random changes ("mutation") such as increasing one type of resource allocation of one job by one unit and decreasing allocation of another job by one unit. After sampling a pre-set number of configurations, GENETIC chooses the configuration with the highest objective function value.
PARTIES. PARTIES [26] is a previously proposed coordinate-descent-like strategy which makes incremental adjustments in one resource at a time until QoS is met for all jobs using finite structure machine decisio-making, or it is deemed that QoS cannot be met for the given configuration.
Brute-Force Search (ORACLE). ORACLE results are obtained offline by sampling every possible configuration and selecting the best one. While, this strategy is infeasible due to the need to sample thousands/millions of configurations, we use it to compare C-LITE against the optimal results.

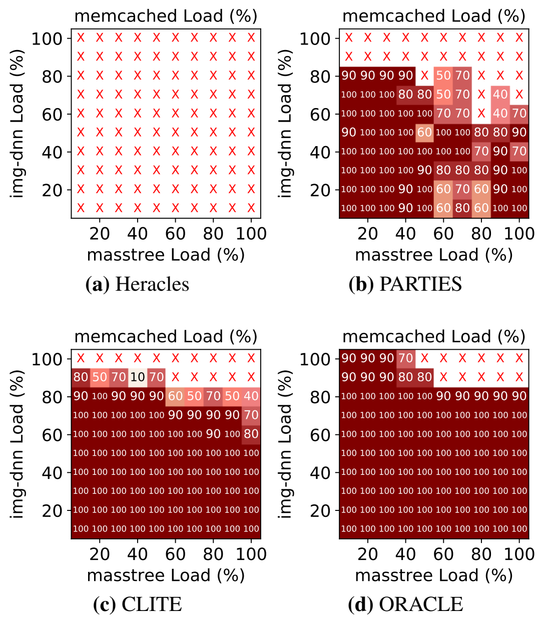


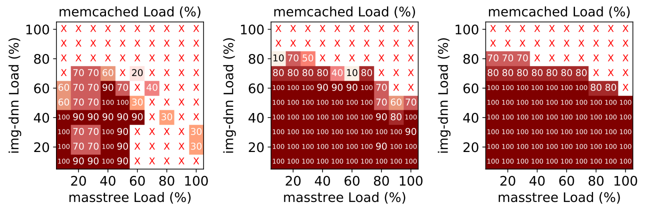
Figure 7: CLITE can co-locate multiple LC jobs together at much higher loads than Heracles and PARTIES, and close to ORACLE. The x- and y- axis show the loads of masstree and img-dnn, respectively, and the plot shows the maximum supported memcached load. X indicates that co-location is not possible.

Finally, we point that we choose different job mixes for different parts of evaluation in order to provide coverage and demonstrate effectiveness of CLITE across diverse scenarios; it is not possible to display all the results for every single job mix, given the space constraints. However, benefits of CLITE across different mixes are statistically similar to ones presented here. Note that for all the heatmaps presented in the following section, higher values (darker colors) are better.

5.2 CLITE: Result Analysis and Discussion

CLITE is efficient in co-locating multiple LC jobs: CLITE can co-locate LC jobs close to ORACLE scheme and at much higher loads compared to Heracles [5] and PARTIES [26]. Fig. 7 shows the highest allowable load for memcached (as % of the maximum load) without violating its QoS for different co-location scheduling policies. In these results, memcached is co-located with two other LC jobs (masstree and img-dnn) at varying loads, without any throughput-oriented BG jobs.

We make three main observations. First, as expected, PARTIES performs better than Heracles. Heracles is not able to



(a) PARTIES **(b)** CLITE **(c)** ORACLE

Figure 8: CLITE can support many more load-configurations than PARTIES, even when running with a BG job (blackscholes) in a 3 LC jobs and 1 BG job mix. The x-axis and y-axis show the loads of corresponding LC jobs, and the plot shows the maximum memcached load which can be supported.

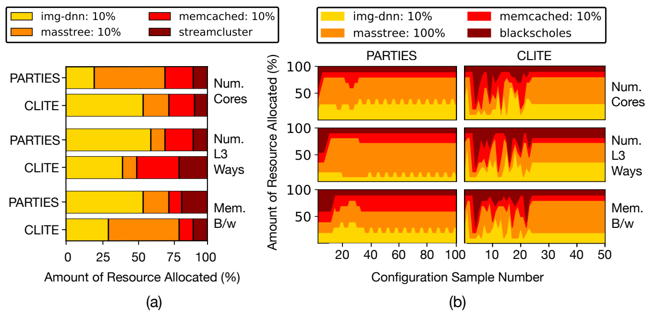


Figure 9: Why does CLITE work better? (a) CLITE explores multiple dimensions simultaneously to achieve more than just meeting QoS, (b) PARTIES cannot co-locate the above set of co-located jobs even after 100 samples, but CLITE can discover near-optimal configuration quickly.

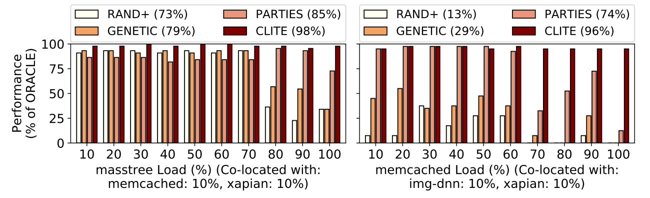


Figure 10: CLITE outperforms w.r.t. the highest mean performance for co-located LC jobs and close to ORACLE (avg. perf. in parenthesis).

co-locate memcached at any load, with 10% loads of img-dnn and masstree. This trend is expected because Heracles is not designed to enable co-location of multiple LC jobs with each achieving its respective QoS. Heracles attempts to provide sufficient resources to meet the QoS of the first LC job. PARTIES overcomes this limitation.

Second, we note that CLITE is able to co-locate more LC jobs than PARTIES. For example, PARTIES cannot co-locate memcached at all, when img-dnn load is 90% and masstree load varies from 10% to 100%. CLITE is able to co-locate memcached even when img-dnn load is 90% and masstree load is below 50%. More importantly, even when PARTIES is able to co-locate memcached, CLITE is always able to co-locate memcached at the same or higher load fraction. Third, as expected, ORACLE scheme performs the best. CLITE performs similar to ORACLE when the load-level of img-dnn is equal to less than 40%, as the load of masstree is varied from 10% to 100%. At very high loads of img-dnn (>70%), CLITE sometimes can co-locate memcached at a lower load than ORACLE. This is because of CLITE's slightly inaccurate BO models - a trade-off made to achieve lower cost.

Fig. 8 shows the highest allowable load for memcached without violating its QoS when co-located with one throughput-oriented BG job (blackscholes) and two other LC jobs (masstree and img-dnn) at varying loads. As expected, compared to Fig. 7, the highest allowable load for the memcached job in Fig. 8 is lower for all schemes (e.g., more boxes with X tick) because of additional co-located BG job. But, as before, CLITE still beats PARTIES by a significant margin (allowing more than 20% load for memcached at high loads of masstree and img-dnn) due to its intelligent resource allocation - this also improves the performance of the throughput-oriented BG jobs (discussed later).

Another interesting property highlighted in Fig. 7 and 8 is the high variability in the highest allowable load for memcached for PARTIES across different configurations (i.e., highest allowable load for the memcached does not decrease continuously in a given row). This variability is caused because of the trial-and-error approach of PARTIES where it explores different combinations in ad-hoc manner and stops when the QoS is met or gives up after certain time. We further quantify this undesirable effect of PARTIES later and compare it with CLITE. Next, we dig deeper using specific examples to understand why CLITE works so effectively.

Why does CLITE work so effectively and perform better than PARTIES? Fig. 9 (a) provides the snapshot of the resource allocation for a particular mix of co-located jobs (img-dnn, memcached, and masstree with streamcluster as the BG job). For this mix, both PARTIES and CLITE attain the QoS targets for all LC jobs, but the resource allocations are different for all jobs. This is because CLITE does not stop after meeting QoS targets, but rather explores space to improve the performance of BG jobs exploiting the "resource equivalence class" property. It takes away particular type of resources from LC jobs to help improve streamcluster performance. CLITE determines these allocations automatically by systematically exploring multiple resource dimensions and applications simultaneously unlike PARTIES. For example, CLITE gives streamcluster more L3 cache ways, while PARTIES gives it more memory bandwidth. But to give streamcluster more L3 cache ways, CLITE had to make completely different allocations for the other three LC jobs. For img-dnn, which performs machine learning inference for image recognition, it is more sensitive on number of cores and L3 cache ways than memory bandwidth. While PARTIES gives it more L3 cache ways, CLITE gives it more cores since it wants to allocate the L3 cache ways to streamcluster. This allocation reduces masstree's core allocation, CLITE gives masstree more memory bandwidth since masstree is sensitive on memory bandwidth. As a result, with CLITE, the BG job, streamcluster's performance is 89% of its ORACLE performance, while, with PARTIES, streamcluster's performance is 39% of ORACLE performance.

Fig. 9 (b) shows the resource allocation over time of a particular load setting where PARTIES does not meet the QoS, while CLITE does. The co-location corresponds to the lower left corner of Fig. 8(a) (img-dnn (LC), memcached (LC), masstree (LC) and blackscholes (BG)). These results provide a deeper view into why trial-and-error approach does not lead to meeting QoS targets even after 100 configuration samples, while CLITE meets the QoS for all three LC jobs in less than

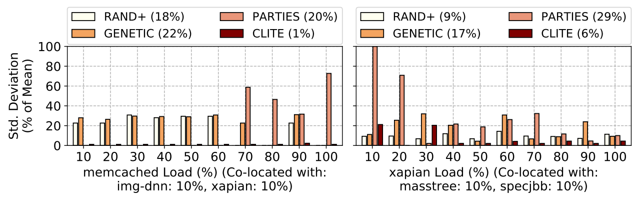


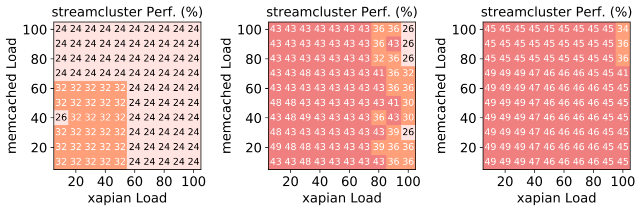
Figure 11: Variability in performances for different trials/runs of the same set of co-located jobs among different comparative techniques (lower is better). CLITE achieves the lowest variability.

30 configuration samples and stabilizes. The reason is the following: initially PARTIES removes all resources from the BG job blackscholes (barring one unit of each), but then it is stuck in cycles in its finite state machine (FSM) [26]. Even through it keeps switching among the different resources due to the FSM structure, it continues to be stuck in cycles and concludes that these LC jobs can not be co-located, while CLITE successfully co-locates them.

CLITE provides high performance to LC jobs when multiple LC jobs are co-located - indicating that shared resources are being utilized efficiently even after the QoSes are met. Fig. 10 shows the average performance for two sets of three co-located LC jobs under different schemes as the load for one of the LC jobs varies and other two are held constant at 10% load. The performance is normalized to the ORACLE scheme which exhaustively tries all the configurations to achieve the highest avg. performance for all three co-located LC jobs while meeting their QoS. We make several interesting observations from these results.

First, on average CLITE performs close to ORACLE and significantly better than competing techniques. For example, CLITE's mean performance across all three co-located LC jobs is 98% and 96% of the ORACLE scheme, while PARTIES is able to provide only 85% and 74%. RAND+ and GA schemes provide performance less than 80% of the ORACLE, although these techniques occasionally beat PARTIES due to their superior evolutionary exploration strategy compared to trial-and-error approach of PARTIES. The reason that CLITE outperforms others is that CLITE is not only focusing on meeting QoS, but also doing resource partitioning to improve performance of each co-located jobs exploiting the "resource equivalence class" property (Sec. 2). CLITE does not stop after meeting QoS targets, it reshuffles resources to improve every job's performance (improving overall resource efficiency of the system). Second, the benefit of CLITE is more pronounced at higher loads of the third co-located job. The is because at higher loads, the competition for resources is higher and the jobs' performances become more sensitive to what type of resources are provided instead of just enough resources to meet the QoS targets.

CLITE not only picks near-optimal configuration, but does so consistently with significantly smaller variability in the final performance. Given that all co-location schemes have stochastic element in partitioning the resources, we investigate how it affects the observed performance of co-located jobs when the exact same set of co-located jobs is run multiple times. Fig. 11 shows the standard deviation (as the % of the mean) of observed performance of co-located LC



(a) PARTIES **(b)** CLITE **(c)** ORACLE

Figure 12: Performance of a BG job (streamcluster) when co-located with memcached and xapian: the x-axis and y-axis show xapian load and memcached load, respectively, and the plot shows the normalized performance of the BG job compared to its isolation performance. CLITE consistently achieves closer performance to ORACLE (darker shade is better).

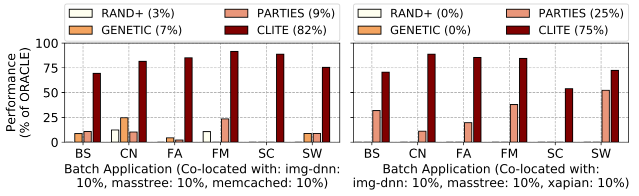


Figure 13: Performance of different BG jobs when running with the different indicated LC job mixes. CLITE achieves much better performance for BG jobs than other techniques. In some cases, other techniques are not even able to find a configuration which meets the QoS of the 3 LC jobs (in which case their BG job performance is marked as 0).

jobs (img-dnn, xapian, & memcached, and specjbb, masstree, & xapian) under different schemes. We observe that the variability in performance of optimal configuration chosen by different schemes across different runs is minimum for CLITE. While the avg. variability in CLITE's optimal configuration is less than 7% in all cases, the corresponding variability for PARTIES, GENETIC, and RAND+ is often more than 20%. The reason behind this higher variability is because competing schemes have a significant factor of randomness (e.g., random configuration in RAND+, probabilistic mutation of configurations in GENETIC, and trial-and-error based reallocation of resources in PARTIES). CLITE has small, albeit non-zero, variability because the dimensionality reduction method choses jobs with a small probabilistic factor.

CLITE partitions the shared resources such that it also provides high performance to throughput-oriented BG jobs while meeting QoS of co-located multiple LC jobs. Fig. 12 shows the performance of the BG job (streamcluster) when co-located with memcached and xapian at different loads for PARTIES, CLITE, and ORACLE (the QoS targets for memcached and xapian is met for all points). The performance of the BG job is normalized w.r.t. their performances in isolation. We notice that CLITE outperforms PARTIES significantly and is within 5% of the ORACLE for most loads.

Next, we evaluate the effectiveness of CLITE for a BG job in the presence of more LC jobs compared to other schemes. Fig. 13 shows the performance of different BG jobs when co-located with different sets of three LC jobs. We make several observations. First, we notice that CLITE provides better performance for all BG jobs compared to all competing schemes, across different sets of LC job mix. On average, CLITE is providing more than 75% of the performance provided by the ORACLE scheme. Second, other competing techniques including PARTIES provide less than 30% of the

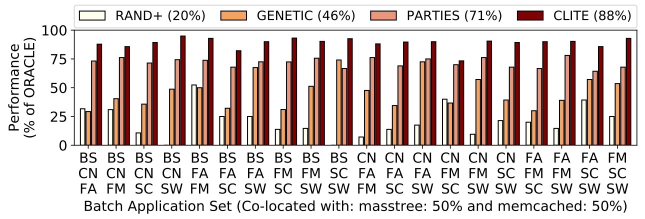


Figure 14: CLITE improves system efficiency by improving BG job performance when multiple BG jobs are co-located with multiple LC jobs. QoS targets of LC jobs are met in all cases. Refer to Table 3 for BG job acronyms.

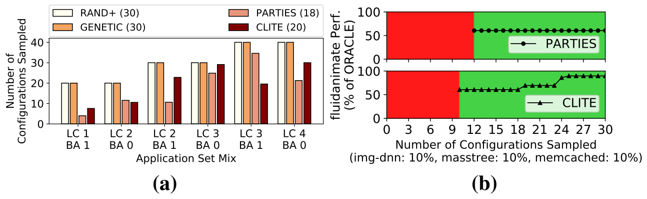


Figure 15: (a) The average overhead of different techniques in terms of the number of configurations sampled, (b) CLITE and PARTIES both find QoS meeting configuration at around the same time, but CLITE keeps improving fluidanimate performance beyond meeting QoS, while PARTIES stabilizes to a suboptimal fluidanimate performance.

performance provided by the ORACLE scheme - more than 40% gap between CLITE and PARTIES - this is because CLITE specifically optimizes for the BG job, while meeting the QoS of all co-located LC jobs.

Even when multiple throughput-oriented BG jobs are co-located with multiple LC jobs, CLITE improves performance of all co-located BG jobs in addition to meeting the QoS targets. One major benefit of CLITE compared to other techniques is that it is multiple-BG-jobs-aware. This allows it to make decisions which cater to not only all the co-located LC jobs but also all the co-located BG jobs. Fig. 14 shows the performances for different multiple-BG-jobs job mixes. Fig. 14 shows that when three BG jobs are co-located with two LC jobs. CLITE performs better by reaching 88% of the optimal performance on average, while the next best technique reaches less than 75% of the optimal on average. The reason is because CLITE's objective function strives to maximize the mean performance of all the co-located BG jobs (as was discussed in Sec. 4 and shown in Eq. 3). It intelligently selects a resource allocation for all BG jobs.

CLITE achieves close-to-optimal performance while incurring only small overhead in sampling configurations - only slightly higher than PARTIES but with much better result quality (i.e., higher degree of co-location and better performance for all co-located jobs). The source of overhead for all co-locating schemes is the sampling of multiple configurations before reaching the best possible configuration. Fig. 15 shows the avg. overhead of all competing co-location scheduling techniques with varying number of LC and BG jobs. Recall ORACLE is an offline technique that exhausts all possible samples to find the optimal - typically 1000s of samples. From Fig. 15 (a), first, we observe that RAND+ and GENTIC have the highest overhead because they collect a pre-set number of samples. This number has been set to be higher than the average number of samples

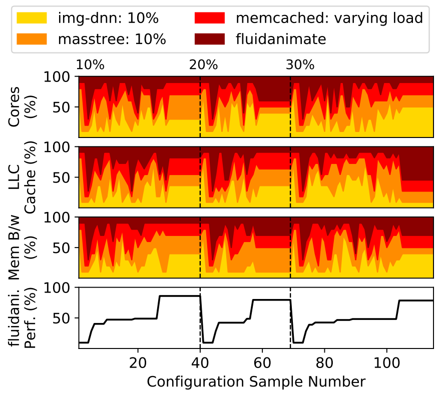


Figure 16: CLITE is adaptive to dynamic load changes. As the memecached load changes from 10% to 30%, CLITE adjusts the resource partition of all co-located jobs and stabilizes to a new optimal resource partitions.

collected by CLITE in order to allow these techniques to be competitive in terms of finding the best configuration even if they have higher overhead. Second, we observe that CLITE typically has slightly higher overhead than PARTIES, but samples only a modest number of samples before reaching its most optimized configuration (less than 30 samples even with high number of co-located jobs) - making it feasible while achieving high quality result. CLITE needs to sample higher number of configurations than PARTIES because PARTIES stops its decision making process as soon as it obtains the QoS-meeting configuration, but CLITE continues to optimize the performance of LC jobs and improve the throughput of BG jobs exploiting resource equivalence class property. Importantly, since PARTIES does not optimize for performance of co-located jobs, its result quality does not improve even if it continues to sample more configurations. Fig. 15(b) shows that CLITE meets the QoS targets for all three co-located LC jobs at the same time or before PARTIES, but continues to sample more to improve the performance of the co-located BG job (fluidanimate). The quality of optimal configuration provided by CLITE improves as more configurations are sampled, but not for PARTIES, since PARTIES samples in an trial-and-error approach without developing a model.

We note that these additional samples compared to PARTIES are the primary overhead of CLITE - it may take slightly longer to converge to near-optimal configuration and hence, falls on the critical path. CLITE's other overhead is related to computation related to decision making and setting the partitions across multiple resources (e.g., taskset, CAT, MBA, etc.). We measured this overhead to be less than 100ms in most cases. But, more importantly, this overhead is not on the critical path and can be overlapped with the evaluation of a previous sample. Therefore, as expected, our experiments confirmed this type of overhead produces negligible effect (less than 0.5%) on QoS quality and performance throughput.

CLITE is adaptive to dynamic load changes, and calculates the new optimal resource partitioning accordingly. CLITE is designed to react appropriately when the load changes and a new resource partitioning needs to be determined to achieve its objectives. Fig. 16 provides the snapshot of the resource allocation for a particular mix of co-located jobs (img-dnn, memcached, and masstree with fluidanimate as the BG job). The load for img-dnn and masstree is fixed at 10%, and the load for memached is changed from 10% to 30% over time. Fig. 16 shows that as the load-level of memcached is increased, CLITE reacts appropriately and determines in a new optimal resource partition for all co-located jobs such that LC jobs meet their QoS requirements and the performance of the BG job is maximized. First, we note that CLITE adjusts the resource partition for different shared resources quickly and stabilizes to a new resource partition configuration. Note that sudden drop in the BG job performance indicates the start of the exploration of the optimal resource partitioning. Second, we note that as the load-level of memcached is increased, the performance of the BG job (fluidanimate) at the stable configuration becomes lower comparatively, over three different periods. This is expected since some resources from the BG job may be allocated to the memcached to meet its QoS at higher load.

CLITE benefits are not sensitive to parameter-tuning of the underlying BO methods. As discussed in Sec. 4, CLITE develops new BO-specific optimizations - some of these require choosing an initial parameter value (e.g., dropout policy and initial configuration space size). As a design principle, we decided to build new strategies to overcome specific challenges instead of focusing on tuning parameters to maximize CLITE's benefits. As a result, we observed that CLITE performs mostly within 2% of the observed performance with reasonably well-chosen parameters without needing excessive tuning. For example, currently, the dropout policy is configured to drop one job at a time during optimization, the number of initial samples is chosen to the number of colocated jobs + 1.

Comparison with design space exploration methods such as Fractional Factorial Designs and Response Surface Methods. We evaluated the suitability of applying design space exploration methods to solve CLITE/PARTIES-like optimization problem. Unfortunately, we found that such methods including Fractional Factorial Designs (FFDs) [55] and Response Surface Methodologies (RSMs) [56], applied previously in computer architecture literature [57], lead to lower quality results even when compared to PARTIES and Genetic Algorithm (GA)-based approaches.

For example, FFDs and RSMs require sampling for more configurations even for simpler cases of CLITE. When designing an FFD for the 2 LC 1 BG scenario (which has 58320 configurations and 9 factors where each factor has 8-9 levels – a large problem generally but a relatively small problem for CLITE), a 2-level FFD needs 48 configurations to sample. On the other hand, if designing an RSM, we need 130 configurations to sample with Box-Behnken RSM and 160 configurations to sample with Central Composite RSM. Already, the number of samples required to be collected is 2x-8x that of CLITE or the other competitive techniques shown in Fig. 15(a). Essentially, these static sampling techniques are not effective because the objective function model varies with the set of co-located jobs, and lack of dynamic and custom sampling (exploration and exploitation) leads to much higher number of samples. These methods are more suitable for relatively smaller problems, as considered in Flicker [57] where the number of factors is 3 while each factor can take on only 3 levels, which results in 27 configurations per core.

Besides choosing a method to decide which configurations to sample, we need to choose the function to fit on to the sample configurations to interpolate the optimal configuration. Traditionally, simple polynomial models are fitted on the response surface, but because of CLITE's complex space with multiple optima (different configurations close-by or far-apart can yield similar results), such models are inadequate. Therefore, we attempted fitting Radial Bias Functions (RBFs) such as polyharmonic spline, inverse multi-quadratics, thin-plate spline - typically used for modeling complex surfaces.

After model fitting and interpolation, we discovered that 2-level FFD is not able to predict the optimal configuration for two LCs to achieve their QoS when co-located with a BG job – a simple example scenario considered in Fig. 12 (memcached: 100% load, xapian: 10% load, and streamcluster). Further, we invested a considerable amount of time to manually tune the parameters of these models to find one resource partitioning scheme that would allow co-location of one particular set of colocated job, but the same set of parameter set did not work as the job mix was changed because the objective function model changes across job sets. In contrast, CLITE's BO based does not require job-set specific parameter tuning and works well across different sets of co-located jobs.

6 Related Work

Co-location works can be broadly categorized as follows:

Unmanaged Co-location Policies. Previous works have proposed solutions for co-locating workloads without any partitioning [58, 1, 10, 59, 8, 11, 12, 60, 13]. The focus is on finding jobs which can be co-located in a manner that they do not contend for the same resources, minimizing their inter-job interference. If the right mix of jobs is found, they can be co-located such that none of their QoSes are violated. Some downsides of these techniques include the requirement to profile jobs beforehand, and the requirement for their dynamic behavior to not change drastically. However, the biggest issue is that the active inter-job interference greatly reduces the types and number of jobs which can be co-located. Other related work include Cooper [61] and Hound [62]. Cooper is a game-theoretic approach for providing fairness and stability guarantees for co-located jobs, and Hound provides a family of techniques to assess and mitigate stragglers in data centers. But these techniques do not provide QoS guarantees for co-located latency-critical jobs.

Partitioning-Based 1-LC/N-BG Co-location Policies. A large number of resource partitioning works have focused on co-locating 1 LC job with BG jobs [14, 5, 15, 16, 17, 63, 18, 64, 65, 66]. These works mainly focus on partitioning to fulfill QoS of the LC job but not on improving the performance of the BG jobs by leveraging resource equivalence. For instance, Dirigent [14] adjusts an LC job's CPU speed and cache partition when it detects that the LC job is violating QoS without considering the BG job's performance. Similarly, [5] focuses on meeting QoS of only 1 LC job, but it does not create resource partitions among the BG jobs, letting them run unmanaged.

Partitioning-Based N-LC/N-BG Co-location Policies. Some partitioning techniques are able to accommodate multiple jobs, but focus on one or two resources [2, 67, 68, 69, 70, 71, 72]. For example, KPart [2] facilitates clustered cache-partitioning, while HyPart [69] facilitates clustered memory bandwidth partitioning. These techniques do not provide frameworks for resource-equivalence-aware partitioning with multiple resources. The only work, to the best of our knowledge, which focuses on co-locating multiple jobs using multi-resource partitioning is PARTIES [26]. However, as shown in Sec. 5, due to its ad-hoc nature, PARTIES is unable to find QoS satisfying configurations in many cases, and does not exploit resource equivalence to maximize the performance of the co-located background jobs.

7 Conclusion

In this paper, we propose CLITE, a Bayesian Optimization-based multi-resource partitioning technique which simultaneously achieves two goals: (1) satisfy QoS requirements of multiple co-located latency-critical jobs, and (2) maximize the performances of multiple co-located batch jobs. Our evaluation shows that CLITE consistently meets QoS for more job sets than can competitive techniques, and it can achieve near-optimal performance for BG jobs. CLITE's low-overhead implementation can greatly help improve co-location efficiency in data centers.

Acknowledgment. We are thankful to anonymous reviewers for their constructive feedback. This work is supported in part by NSF Awards 1910601 and 1753840, Northeastern University, and Massachusetts Green High Performance Computing Center (MGHPCC).

8 References

[1] H. Yang, A. Breslow, J. Mars, and L. Tang, "Bubble-Flux: Precise Online QoS Management for Increased Utilization in Warehouse Scale Computers," *ACM Computer Architecture News*, vol. 41, no. 3, pp. 607–618, 2013.

[2] N. El-Sayed, A. Mukkara, P.-A. Tsai, H. Kasture, X. Ma, and D. Sanchez, "KPart: A Hybrid Cache Partitioning-Sharing Technique for Commodity Multicores," in *2018 IEEE HPCA*, pp. 104–117, 2018.

[3] J. Chen and L. K. John, "Predictive Coordination of Multiple On-Chip Resources for Chip Multiprocessors," in *Proceedings of Supercomputing*, pp. 192–201, ACM, 2011.

[4] P. Thinakaran, J. R. Gunasekaran, B. Sharma, M. T. Kandemir, and C. R. Das, "Phoenix: A Constraint-Aware Scheduler for Heterogeneous Datacenters," in *2017 IEEE 37th Conf. on Distributed Computing Systems (ICDCS)*, pp. 977–987, IEEE, 2017.

[5] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: Improving Resource Efficiency at Scale," in *ACM Computer Architecture News*, vol. 43, pp. 450–462, ACM, 2015.

[6] L. He, S. Yao, and W. Zhou, "An Extended Fine-Grained Conflict Detection Method for Shared-State Scheduling in Large Scale Cluster," in *2016 Conf. on Intelligent Information Processing*, p. 29, ACM, 2016.

[7] L. Tang, J. Mars, W. Wang, T. Dey, and M. L. Soffa, "Reqos: Reactive static/dynamic compilation for qos in warehouse-scale computers," in *ACM Computer Architecture News*, vol. 41, pp. 89–100, ACM, 2013.

[8] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-Locations," in *Proceedings of IEEE/ACM Symp. on Microarchitecture*, pp. 248–259, ACM, 2011.

[9] L. Tang, J. Mars, and M. L. Soffa, "Compiling for Niceness: Mitigating Contention for QoS in Warehouse Scale Computers," in *Tenth Symp. on Code Generation and Optimization*, pp. 1–12, ACM, 2012.

[10] Y. Zhang, M. A. Laurenzano, J. Mars, and L. Tang, "SMiTe: Precise QoS Prediction on Real-System SMT Processors to Improve Utilization in Warehouse Scale Computers," in *2014 47th IEEE/ACM Symp. on Microarchitecture*, pp. 406–418, IEEE, 2014.

[11] S. Blagodurov, A. Fedorova, E. Vinnik, T. Dwyer, and F. Hermenier, "Multi-Objective Job Placement in Clusters," in *SC'15: Conf. for High Performance Computing, Networking, Storage and Analysis*, pp. 1–12, IEEE, 2015.

[12] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: Fair Scheduling for Distributed Computing Clusters," in *ACM SIGOPS 22nd Symp. on Operating systems principles*, pp. 261–276, ACM, 2009.

[13] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-Efficient and QoS-Aware Cluster Management," in *ACM Computer Architecture News*, vol. 42, pp. 127–144, ACM, 2014.

[14] H. Zhu and M. Erez, "Dirigent: Enforcing QoS for Latency-Critical Tasks on Shared Multicore Systems," *ACM Computer Architecture News*, vol. 44, no. 2, pp. 33–47, 2016.

[15] H. Kasture, D. B. Bartolini, N. Beckmann, and D. Sanchez, "Rubik: Fast Analytical Power Management for Latency-Critical Systems," in *IEEE/ACM MICRO*, pp. 598–610, IEEE, 2015.

[16] X. Wang, S. Chen, J. Setter, and J. F. Martínez, "SWAP: Effective Fine-Grain Management of Shared Last-Level Caches with Minimum Hardware Support," in *2017 IEEE Symp. on High Performance Computer Architecture (HPCA)*, pp. 121–132, IEEE, 2017.

[17] D. Sanchez and C. Kozyrakis, "Vantage: Scalable and Efficient Fine-Grain Cache Partitioning," in *ACM Computer Architecture News*, vol. 39, pp. 57–68, ACM, 2011.

[18] H. Kasture and D. Sanchez, "Ubik: Efficient Cache Sharing with Strict QoS for Latency-Critical Workloads," in *ACM Computer Architecture News*, vol. 42, pp. 729–742, ACM, 2014.

[19] R. S. Kannan, L. Subramanian, A. Raju, J. Ahn, J. Mars, and L. Tang, "GrandSLAm: Guaranteeing SLAs for Jobs in Microservices Execution Frameworks," in *Fourteenth EuroSys Conf. 2019*, p. 34, ACM, 2019.

[20] A. Mirhosseini and T. F. Wenisch, "The Queuing-First Approach for Tail Management of Interactive Services," *IEEE Micro*, vol. 39, no. 4, pp. 55–64, 2019.

[21] A. Sriraman, A. Dhanotia, and T. F. Wenisch, "Softsku: Optimizing Server Architectures for Microservice Diversity at Scale," in *46th Symp. on Computer Architecture*, pp. 513–526, ACM, 2019.

[22] J. Thönes, "Microservices," *IEEE software*, vol. 32, no. 1, pp. 116–116, 2015.

[23] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson, *et al.*, "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems," in *Twenty-Fourth Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 3–18, ACM, 2019.

[24] Y. Gan, Y. Zhang, K. Hu, D. Cheng, Y. He, M. Pancholi, and C. Delimitrou, "Seer: Leveraging Big Data to Navigate the Complexity of Performance Debugging in Cloud Microservices," in *Twenty-Fourth Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 19–33, ACM, 2019.

[25] Y. Gan and C. Delimitrou, "The Architectural Implications of Cloud Microservices," *IEEE Computer Architecture Letters*, vol. 17, no. 2, pp. 155–158, 2018.

[26] S. Chen, C. Delimitrou, and J. F. Martínez, "PARTIES: QoS-Aware Resource Partitioning for Multiple Interactive Services," in *Twenty-Fourth Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 107–120, ACM, 2019.

[27] R. S. Kannan, M. Laurenzano, J. Ahn, J. Mars, and L. Tang, "Caliper: Interference Estimator for Multi-tenant Environments Sharing Architectural Resources," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 16, no. 3, p. 22, 2019.

[28] C. Witt, M. Bux, W. Gusew, and U. Leser, "Predictive Performance Modeling for Distributed Batch Processing using Black Box Monitoring and Machine Learning," *Information Systems*, 2019.

[29] S. A. Javadi, A. Suresh, M. Wajahat, and A. Gandhi, "Scavenger: A Black-Box Batch Workload Resource Manager for Improving Utilization in Cloud Environments," in *ACM Symp. on Cloud Computing*, pp. 272–285, ACM, 2019.

[30] C. Intel, “Improving real-time performance by utilizing cache allocation technology,” *Intel Corporation, April*, 2015.

[31] “Intel 64 and IA-32 Architectures Software Developer’s Manual.”

[32] D. Shen, Q. Luo, D. Poshyvanyk, and M. Grechanik, “Automating performance bottleneck detection using search-based application profiling,” in *2015 Symp. on Software Testing and Analysis*, pp. 270–281, ACM, 2015.

[33] S.-H. Lim, J.-S. Huh, Y. Kim, G. M. Shipman, and C. R. Das, “D-Factor: A Quantitative Model of Application Slow-Down in Multi-Resource Shared Systems,” *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 1, pp. 271–282, 2012.

[34] S. Ruder, “An Overview of Gradient Descent Optimization Algorithms,” *arXiv preprint arXiv:1609.04747*, 2016.

[35] P. Netrapalli, “Stochastic Gradient Descent and Its Variants in Machine Learning,” *Journal of the Indian Institute of Science*, pp. 1–13, 2019.

[36] X. Lin and G. Wei, “Generalized Non-Convex Non-Smooth Sparse and Low Rank Minimization Using Proximal Average,” *Neurocomputing*, vol. 174, pp. 1116–1124, 2016.

[37] A. Scotto Di Perrotolo, “A Theoretical Framework for Bayesian Optimization Convergence,” 2018.

[38] T. Miyazaki, I. Sato, and N. Shimizu, “Bayesian Optimization of HPC Systems for Energy Efficiency,” in *Conf. on High Performance Computing*, pp. 44–62, Springer, 2018.

[39] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, “Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics,” in *14th {USENIX} Symp. on Networked Systems Design and Implementation ({NSDI} 17)*, pp. 469–482, 2017.

[40] V. Nair, R. Krishna, T. Menzies, and P. Jamshidi, “Transfer Learning with Bellwethers to Find Good Configurations,” *arXiv preprint arXiv:1803.03900*, 2018.

[41] C.-J. Hsu, V. Nair, T. Menzies, and V. Freeh, “Micky: A Cheaper Alternative for Selecting Cloud Instances,” in *2018 IEEE 11th Conf. on Cloud Computing (CLOUD)*, pp. 409–416, IEEE, 2018.

[42] C.-J. Hsu, V. Nair, T. Menzies, and V. W. Freeh, “Scout: An Experienced Guide to Find the Best Cloud Configuration,” *arXiv preprint arXiv:1803.01296*, 2018.

[43] J. Snoek, H. Larochelle, and R. P. Adams, “Practical Bayesian Optimization of Machine Learning Algorithms,” in *Advances in neural information processing systems*, pp. 2951–2959, 2012.

[44] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, *et al.*, “Taking the Human Out of the Loop: A Review of Bayesian Optimization,” *Proceedings of IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

[45] K. Kawaguchi, L. P. Kaelbling, and T. Lozano-Pérez, “Bayesian Optimization with Exponential Convergence,” in *Advances in neural information processing systems*, pp. 2809–2817, 2015.

[46] D. J. Lizotte, *Practical Bayesian Optimization*. Uni. of Alberta, 2008.

[47] C. Li, S. Gupta, S. Rana, V. Nguyen, S. Venkatesh, and A. Shilton, “High Dimensional Bayesian Optimization using Dropout,” *arXiv preprint arXiv:1802.05400*, 2018.

[48] K. Kandasamy, K. R. Vysyaraju, W. Neiswanger, B. Paria, C. R. Collins, J. Schneider, B. Poczos, and E. P. Xing, “Tuning Hyperparameters without Grad Students: Scalable and Robust Bayesian Optimisation with Dragonfly,” *arXiv preprint arXiv:1903.06694*, 2019.

[49] B. Letham, B. Karrer, G. Ottoni, E. Bakshy, *et al.*, “Constrained Bayesian Optimization with Noisy Experiments,” *Bayesian Analysis*, vol. 14, no. 2, pp. 495–519, 2019.

[50] H. Kasture and D. Sanchez, “Tailbench: A Benchmark Suite and Evaluation Methodology for Latency-Critical Applications,” in *2016 IEEE Symp. on Workload Characterization (IISWC)*, pp. 1–10, IEEE, 2016.

[51] C. Bienia and K. Li, “Parsec 2.0: A New Benchmark Suite for Chip-Multiprocessors,” in *5th Workshop on Modeling, Benchmarking and Simulation*, vol. 2011, 2009.

[52] N. Kulkarni, F. Qi, and C. Delimitrou, “Pliant: Leveraging Approximation to Improve Datacenter Resource Efficiency,” in *2019 IEEE Symp. on High Performance Computer Architecture (HPCA)*, pp. 159–171, IEEE, 2019.

[53] B. Fitzpatrick, “Distributed Caching with Memcached,” *Linux journal*, vol. 2004, no. 124, p. 5, 2004.

[54] J. Leverich, “Mutilate: High-Performance Memcached Load Generator,” 2014.

[55] R. F. Gunst and R. L. Mason, “Fractional factorial design,” *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 1, no. 2, pp. 234–244, 2009.

[56] F. Azadivar, “Simulation optimization methodologies,” in *WSC’99. 1999 Winter Simulation Conf. Proceedings.’Simulation-A Bridge to the Future’(Cat. No. 99CH37038)*, vol. 1, pp. 93–100, IEEE, 1999.

[57] P. Petrica, A. M. Izraelevitz, D. H. Albonesi, and C. A. Shoemaker, “Flicker: A dynamically adaptive architecture for power limited multicore systems,” in *ACM Computer Architecture News*, vol. 41, pp. 13–23, ACM, 2013.

[58] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes, “CPI2: CPU Performance Isolation for Shared Compute Clusters,” in *8th ACM European Conf. on Computer Systems*, pp. 379–391, ACM, 2013.

[59] C. Delimitrou and C. Kozyrakis, “QoS-Aware Scheduling in Heterogeneous Datacenters with Paragon,” *ACM Transactions on Computer Systems (TOCS)*, vol. 31, no. 4, p. 12, 2013.

[60] C. Delimitrou, D. Sanchez, and C. Kozyrakis, “Tarcil: Reconciling Scheduling Speed and Quality in Large Shared Clusters,” in *Sixth ACM Symp. on Cloud Computing*, pp. 97–110, ACM, 2015.

[61] Q. Llull, S. Fan, S. M. Zahedi, and B. C. Lee, “Cooper: Task colocation with cooperative games,” in *2017 IEEE Symp. on High Performance Computer Architecture (HPCA)*, pp. 421–432, IEEE, 2017.

[62] P. Zheng and B. C. Lee, “Hound: Causal learning for datacenter-scale straggler diagnosis,” *ACM on Measurement and Analysis of Computing Systems*, vol. 2, no. 1, p. 17, 2018.

[63] C.-J. Wu and M. Martonosi, “A Comparison of Capacity Management Schemes for Shared CMP Caches,” in *Proc. of the 7th Workshop on Duplicating, Deconstructing, and Debunking*, vol. 15, pp. 50–52, Citeseer, 2008.

[64] Q. Chen, Z. Wang, J. Leng, C. Li, W. Zheng, and M. Guo, “Avalon: Towards QoS Awareness and Improved Utilization Through Multi-Resource Management in Datacenters,” in *ACM Conf. on Supercomputing*, pp. 272–283, ACM, 2019.

[65] W. Zhang, W. Cui, K. Fu, Q. Chen, D. E. Mawhirter, B. Wu, C. Li, and M. Guo, “Laius: Towards Latency Awareness and Improved Utilization of Spatial Multitasking Accelerators in Datacenters,” in *ACM Conf. on Supercomputing*, pp. 58–68, ACM, 2019.

[66] C. Iorgulescu, R. Azimi, Y. Kwon, S. Elnikety, M. Syamala, V. Narasayya, H. Herodotou, P. Tomita, A. Chen, J. Zhang, *et al.*, “PerfIso: Performance Isolation for Commercial Latency-Sensitive Services,” in *2018 {USENIX} Technical Conf. ({USENIX}{ATC} 18)*, pp. 519–532, 2018.

[67] Y. Xiang, X. Wang, Z. Huang, Z. Wang, Y. Luo, and Z. Wang, “DCAPS: Dynamic Cache Allocation with Partial Sharing,” in *Thirteenth EuroSys Conference*, p. 13, ACM, 2018.

[68] C. Xu, K. Rajamani, A. Ferreira, W. Felter, J. Rubio, and Y. Li, “dCat: Dynamic Cache Management for Efficient, Performance-Sensitive Infrastructure-as-a-Service,” in *Thirteenth EuroSys Conference*, p. 14, ACM, 2018.

[69] J. Park, S. Park, M. Han, J. Hyun, and W. Baek, “HyPart: A Hybrid Technique for Practical Memory Bandwidth Partitioning on Commodity Servers,” in *27th Conf. on Parallel Architectures and Compilation Techniques*, p. 5, ACM, 2018.

[70] J. Park, S. Park, and W. Baek, “CoPart: Coordinated Partitioning of Last-Level Cache and Memory Bandwidth for Fairness-Aware Workload Consolidation on Commodity Servers,” in *Fourteenth EuroSys Conf. 2019*, p. 10, ACM, 2019.

[71] S. Srikantaiah, R. Das, A. K. Mishra, C. R. Das, and M. Kandemir, “A Case for Integrated Processor-Cache Partitioning in Chip Multiprocessors,” in *Conf. on High Performance Computing Networking, Storage and Analysis*, p. 6, ACM, 2009.

[72] A. Margaritov, S. Gupta, R. Gonzalez-Alberquilla, and B. Grot, “Stretch: Balancing QoS and Throughput for Colocated Server Workloads on SMT Cores,” in *2019 IEEE Symp. on High Performance Computer Architecture (HPCA)*, pp. 15–27, IEEE, 2019.