High-Dimensional Bayesian Optimization with Multi-Task Learning for RocksDB

Sami Alabed
sa894@cam.ac.uk
University of Cambridge
Cambridge, UK

Eiko Yoneki
eiko.yoneki@cl.cam.ac.uk
University of Cambridge
Cambridge, UK

Abstract

RocksDB is a general-purpose embedded key-value store used in multiple different settings. Its versatility comes at the cost of complex tuning configurations. This paper investigates maximizing the throughput of RocksDB IO operations by auto-tuning ten parameters of varying ranges. Off-the-shelf optimizers struggle with high-dimensional problem spaces and require a large number of training samples.

We propose two techniques to tackle this problem: multi-task modeling and dimensionality reduction through clustering. By incorporating adjacent optimization in the model, the model converged faster and found complicated settings that other tuners could not find. This approach had an additional computational complexity overhead, which we mitigated by manually assigning parameters to each sub-goal through our knowledge of RocksDB. The model is then incorporated in a standard Bayesian Optimization loop to find parameters that maximize RocksDB's IO throughput.

Our method achieved x1.3 improvement when benchmarked against a simulation of Facebook's social graph traffic, and converged in ten optimization steps compared to other state-of-the-art methods that required fifty steps.

CCS Concepts: • **Information systems → Key-value stores**; • **Computing methodologies → Gaussian processes**.

Keywords: Multi-Task Learning, Bayesian Optimization

1 Introduction

A high-dimensional optimization space is a common phenomenon in general-purpose systems as they have many parameters and objectives. Here we investigate the optimization of RocksDB, a popular key-value store and the building block of many mission-critical systems. For example, Facebook uses RocksDB as a backend to diverse systems with high-query, high-throughput, or low-latency requirements [8]. These widely different use-cases have proved difficult to design a one-size fit all system due to these challenges:

- RocksDB [16] has over thirty parameters. The sheer number of configurations is overwhelming to the user.
- The hardware directly impacts the user's choice of configurations. For example, configurations that are valid on SSD perform suboptimally on HDD [33].
- Each application has unique access patterns [8].

These challenges motivate the need for auto-tuning methods. Methods such as random search [5], hill-climbing [2], reinforcement learning [39], or population-based search [22] find optimal configurations in high-dimensions. However, they suffer from requiring many training samples that are expensive to come by. Every training sample requires a full instance restart and execution of a resource-intensive benchmark.

Bayesian Optimization (BO) [38], a sample efficent tuner, has received considerable attention in recent years due to its versatility and efficiency. The framework, illustrated in Figure 1, first builds a system model and then uses the model to find an optimal configuration, thus reducing the interaction with RocksDB and leading to fewer evaluations.
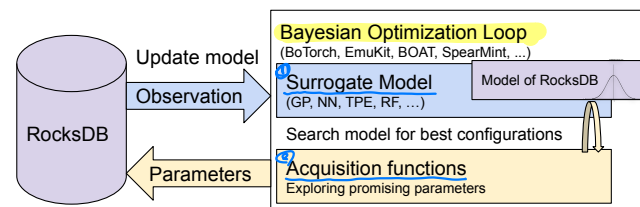

Figure 1. Bayesian optimization training loop. A surrogate model builds knowledge of the system. The acquisition function finds the most promising configurations in the model to try on the real system.
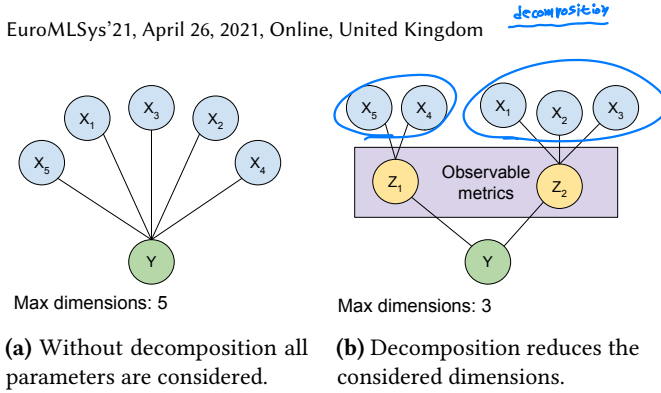
(a) Without decomposition all parameters are considered.

(b) Decomposition reduces the considered dimensions.

Figure 2. Illustration of structural decomposability. Structural modeling reduces the dimensional space of the problem. Figure 2b the maximum effective dimension is three for the the metric with largest number of parameters: $p(z_2|x_1, x_2, x_3)$.

BO's drawback is its inability to handle high-dimensional spaces due to the *curse of dimensionality* [41] and a computationally expensive operation in the surrogate model. In this paper, we mitigate these drawbacks by adding expert knowledge to the BO surrogate model that exploits the system's properties. This work addresses the high-dimensionality problem with two novel ideas: optimizing over multiple targets to increase learning mileage per training sample; and sub-task decomposition to reduce the problem's dimensions.

Optimizing over multiple targets is especially beneficial to computer systems as their performance depends on various internal components at work. For example, in the RocksDB case, the write throughput is bottlenecked by IO stalls, reducing the time spent stalling improves the throughput. This knowledge provides the model with a broader overview of the system's behavior.

Adding optimization targets has an increased computational cost. Hence the second part of this work mitigates this cost through decomposition. Figure 2 demonstrates structural decomposability: rather than jointly considering all parameters and assuming they all influence the main optimization target, the primary optimization target decomposes into a subset of system components that influence the primary target.

We position our work as a contribution towards utilizing computer systems structure in an accessible way and highlight it in a case-study of maximizing RocksDB's IO throughputs on Facebook's social graph workload [8].

In summary, our main contributions are the following:

- A method to inject structural knowledge in a BO optimizer significantly reducing the convergence time from seventy to ten steps.
- The reduction of the problem's dimensionality using manual task decomposition.
- An IO operations throughput improvement of up to x1.45 over RocksDB's default configurations.

2 Background

2.1 RocksDB

RocksDB is a key-value store based on LevelDB [13] that provides efficient concurrent reads and writes. RocksDB stores new data in a log-structured merge-tree format [28] (in memory (memtable)). Once the memtable is full, RocksDB flushes it sequentially into a Sorted Sequence Tree (SST) file on disk for persistent storage. RocksDB still processes concurrent writes during the flushing process; hence, it is a popular choice for high-throughput applications such as stream processors [9].

SST organizes the data in levels starting from level-0 to level-n. When a level is full, it performs a housekeeping operation (compaction) that merges levels and removes tombstones. During compaction, RocksDB stalls new writes, leading to a reduced IO throughput and increased latency, which is why each level's size is an important parameter. A larger level size reduces compactions frequency; however, read operations will take longer to perform as there are more SST to scan before finding keys. Readers are referred to [14, 17] for a detailed information about RocksDB architecture.

2.2 Bayesian optimization

BO is a sample efficient optimization framework [38] that solves the problem of black-box function optimization. It performs a series of iterative operations that builds knowledge of the function and then optimizes it. Formally, it is expressed as $x^* = argmax(f(x))$, where f is the objective function that typically takes $x \in R^D$ as a parameter, and D its dimensionality.

BO's algorithm shown in 1 has two main components: a surrogate model and an acquisition function. The surrogate model contains a belief of the system and updates at every new observation. The acquisition function performs numerical optimization operations over the model to find configurations to test next on the objective function. It aims to balance the trade-off between exploring and exploitation. Exploring the system provides more information for the surrogate model while exploiting it to meet the primary optimization target. Shahriari et al. [36] provides an in-depth survey of BO and its components.

Algorithm 1 Bayesian optimization

Input: acquisition function α, surrogate model M

    Initialize a list D holding the observed data
    while there is budget **do**
        select next point x_{n+1} to evaluate
        $x_{n+1} = \alpha(M, D)$
        submit x_{n+1} to be evaluated next and observe results
        $y_{n+1} = system(x_{n+1})$
        augment the observed data with a new observation
        $D \cup (x_{n+1}, y_{n+1})$
    end while

2.3 Surrogate model

2.3.1 Gaussian Process (GP).

A GP [32] is a popular and powerful non-parametric model that captures the relation between complex variables in a multivariate normal distribution. The GP's ability to regularize and inject priors over the objective function space allows it to capture complex models efficiently.

Formally expressed as $GP(\mu, K)$ where μ is the mean function, often set to a constant value (zero or the mean of the data) and K is a covariance kernel. While GP is powerful and sample efficient, they struggle in high-dimensional settings $x \in R^D, D \geq 10$ [42]. That is due to two main reasons: firstly, the GP analytical inference is computationally expensive, as it performs a $O(N^3)$ matrix inversion operation to calculate the log-likelihood; secondly, when the dimensionality increases, every training sample provide less information due to the *curse of dimensionality* [41]. Furthermore, a fundamental assumption of a GP is that the parameter space form a multivariate Gaussian distribution.

Previous work has reduced the complexity of the GP. For example, GPyTorch [18] uses efficient operations in a matrix-to-matrix structure to improve the average complexity to $O(N^2)$ when there is a large number of training samples. Other methods use approximate inference by backpropagating gradients and performing variational inference [20], these methods are highly scalable and benefit from the Autograd optimizations provided by PyTorch and TensorFlow. However, firstly, they require more training samples as they approximate the GP to the nearest Gaussian; secondly, they lack eventual convergence guarantees.

2.3.2 Alternative surrogate models.

BO's framework is modular and allows many other surrogate models. We list some of the popular ones here:

- Additive kernels [15] decompose a function into a sum of independent low-dimensional kernels operating on a subset of the input space, allowing the model to handle high dimensional parameter space.
- Tree-structured Parzen Estimator (TPE) model [4] is designed to handle discrete variables out-of-the-box. It collects and scores previously observed parameter space into two kernel estimator densities. One is the most performant observation, and the other branch is everything else. The densities act as the surrogate model. TPE scales linearly with the data and handles categorical parameters but is less sample efficient than GPs [38].
- Random forest (RF) [24] is a model that linearly scales with the data and handles any parameter type. The acquisition function requires an uncertainty estimate from the surrogate model to perform exploration over its space. HyperMapper [26] and SMAC [21] use RF as a surrogate model by providing the empirical variance as an uncertainty estimate.

3 Structured multi-task optimization

3.1 Overview

Reducing the number of steps the tuner performs for finding optimal settings in RocksDB leads to considerable costs saving; hence it is the primary focus of this work.

As the previous section discussed, Bayesian optimization (BO) is a samples-efficient framework that has found successes in tuning systems. Injecting experts' knowledge in BO is the subject of several interesting previous works [11, 26, 27]. While very performant and offering great flexibility, these methods have a steep learning curve.

This work utilizes an alternative mechanism to inject expert knowledge into the model; the user identifies low-level metrics to optimize that also improve their main objective. We used multi-task learning to capture the interaction between system components and learn more from every sample, reducing the observations needed for convergence. We presented an extension that reduces the dimensions through a manual grouping of parameters to speed up the convergence by reducing the maximum dimension.

3.2 Problem space and assumptions

The fundamental assumption in using GP is that the modeling space is a multivariate Gaussian distribution, and the function is differentiable at every point (smooth). We performed 500 independent experiments where we randomly sampled the modeling space (ten parameters) to learn if a GP can model the configurations' impact on the system.

Figure 3, shows a histogram of RocksDB's IO throughput (IOPS) when randomly sampling the selected ten parameters. The histogram shows a smooth distribution that a GP can model. We do not consider categorical parameters as they cause jumps in the observed distribution, violating the smoothness assumption; this is a known issue for GP models and outside our scope.
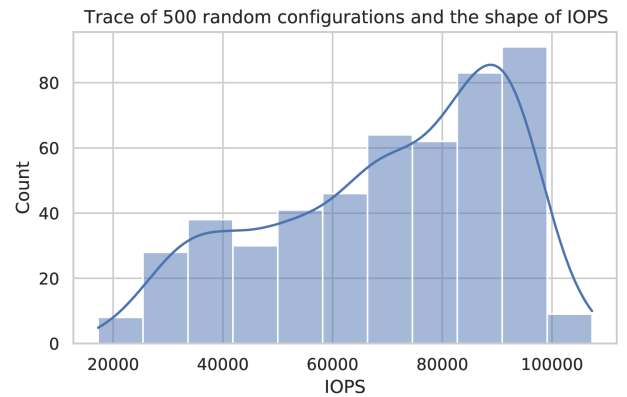

Figure 3. Histogram of RocksDB's IO throughput when changing parameters randomly in 500 samples is a smooth function that a Gaussian Process can approximate.
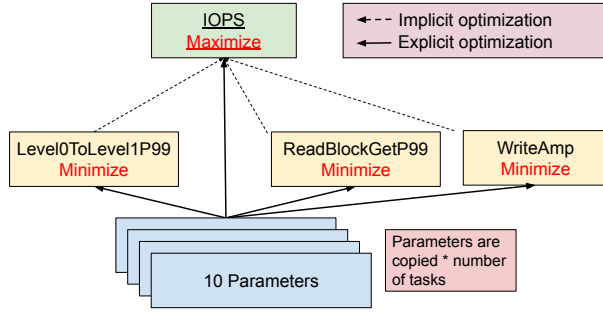
Figure 4. A high-level view of multi-task optimization benefit in RocksDB. Optimizing the complimentary tasks provide implicit optimization to the primary goal, leading to TASK number of information per training sample.

3.3 Multi-task learning

3.3.1 Tasks. This paper's thesis is that the main objective, IOPS, can be implicitly improved by optimizing adjacent objectives. We chose three additional objectives based on our understanding of RocksDB architecture. Each of these objectives directly or indirectly impacts IOPS as illustrated in Figure 4. These objectives each measures the performance of a separate component of RocksDB:

- **WriteAmplification**: the ratio of bytes written to storage to the bytes written to the backend. Higher amplification reduces IOPS as it causes larger writes to disk and gets bottlenecked by the disk write speed.
- **ReadBlockGetP99**: The $99th$ percentile latency to read a block of data. This target forces the optimizer to balance improving write throughput against read throughput, which provides a better IOPS.
- **Level0Tolevel1P99**: The $99th$ percentile time it takes to compact blocks stored in level0 to level1. IO stalls while the system is compacting the blocks. Thus this encodes the tradeoff between increased IOPS by using larger memory against having to stall later while compacting blocks out of the memory once it is full. Level0To1 is the most important one, as higher levels are stored on disk and larger, so they happen less regularly, while level0 flushes happen very frequently.

3.3.2 Intrinsic coregionalization model. Modeling the three additional tasks requires changes to the kernel function of the GP. The popular method to enable multi-task learning in GP uses the Intrinsic Coregionalization Model (ICM) [1] kernel. We used GPyTorch [18] implementation of ICM kernel. It is expressed in Equation 1, where $k_x(x, x')$ is the parameter covariance kernel, and $k_T(m, m')$ is the task similarity kernel.

$$k((x, m), (x', m')) = k_x(x, x')k_T(m, m') \tag{1}$$

Using this kernel in the GP outputs a multivariate Gaussian distribution for each of the tasks. Each output is then optimized jointly as part of the BO loop.

The ICM kernel is an excellent tool to learn implicit system structure as it learns two things: the covariance between configurations and observation, and a task similarity matrix for sharing knowledge between tasks. The ability to share knowledge between tasks and provide $(TASKS * N)$ additional training samples for the model to utilize makes ICM suitable for improving its convergence speed.

Algorithm 2 shows the preprocessing steps to using ICM for RocksDB tuning. We scale every historical configuration used to tune RocksDB by the number of tasks available and normalize the configurations into a unit cube $[0, 1]^D$, so the model considers all parameters equally. Next, tasks are standardized around zero mean and unit variance; this smoothens the task distribution's shape providing an easier target for the GP to model.

Algorithm 2 RocksDB Multi-Task Model

Input: Observed metrics list T and previous parameters X

Initialize D linking parameters to their associated task.
Normalize all previous observations X to the unit cube.
$\forall x \in X \Rightarrow x' = \frac{x - min(x)}{max(x) - min(x)}$
for each *task* t in T **do**
    Standardize the task to a zero mean and a unit variance.
    $y_t = \frac{(task) - mean(task)}{\sigma}$
    Assign previous observation to each task .
    $D \cup (x, y_t)$
end for
Model all the tasks using ICM kernel K
return $GP(D)$

3.3.3 ICM challenges. ICM method provides a neat trick to get more mileage out of the few samples we usually have when tuning computer systems. However, it raises some scalability concerns; a standard GP inference is $O(n^3)$, duplicating the data to the number of tasks scales this to $O(Tn^3)$ which can be troublesome. While it can be argued that computer systems rarely have enough samples to reach that bottleneck, if we look into optimizing more than ten parameters, this will hit a wall very quickly. This issue motivates the next subsection on decomposition.

3.4 Decomposability through clustering

3.4.1 Decomposability in RocksDB. The large parameter space in RocksDB is a challenge for most optimizers due to the *curse of dimensionality* [41] and the high computational cost. Using the multi-task method mitigates some of the first issues at an increased computational cost.

To reduce the computational cost, we exploit a fundamental property in RocksDB design; the final observed metric is the sum of multiple internal components performance; this is known in the literature as functional decomposability [7]. The decomposability refers to the smallest unit of observable RocksDB's performance metric and a corresponding set of
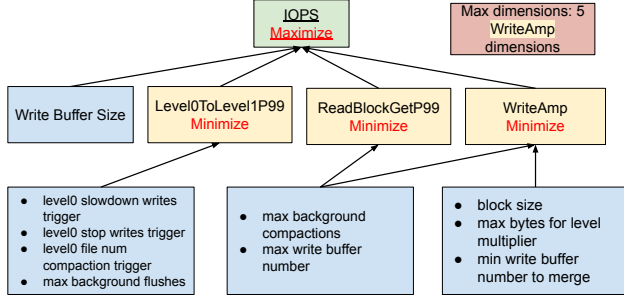
Figure 5. We manually assign a subset of parameters to each optimization target, thus reducing the dimensions. The assignment was done by picking association with the highest covariance values.

parameters in this context. By decomposing it, we reduce the effective dimensions as each model is self-contained, and thus the joint probability distribution of that model is much smaller.

3.4.2 Manual clustering. As the modeling complexity linked directly to the dimension of the parameter space, by slicing the dimensions, we get a significant boost to both inference time and reducing the impact of *curse of dimensionality*. Using the 500 random configurations trace, we calculated the correlations between IOPS and the 517 observable metric from RocksDB and the correlations between them to the parameters. This reduced the pool of candidate clusters to 30, of which we applied expert knowledge and performing isolated experiments to reach the structure shown in the figure.

We decompose the parameter space and use independent GPs for each cluster of parameters. Some of these GPs use the ICM kernel to share knowledge between the related subset of tasks and parameters. Figure 5 illustrates the main idea behind this method. This method provides the best of both worlds: reduced dimensionality from clustering and increased system knowledge from the shared kernel.

3.4.3 Unsupervised clustering. We recognize the opportunity to employ unsupervised clustering to find these clusters. An unsupervised approach provides the most straightforward interface for the end-user; the user provides the various metrics to the tuner, while the tuner takes on the responsibility to find structure in the system using these system observations. The unsupervised approach combined with multi-task learning allows for learning in situations where a parameter is assigned to multiple tasks simultaneously. A parameter impacting multiple different clusters can often happen in RocksDB. For example, the buffer size has a positive correlation to latency but a negative correlation to compaction time. We leave this as future work and show our preliminary results here.

Table 1. RocksDB parameters and their impact. All reported parameters are discrete ordinal variables.

Parameter	Range	Default
max_background_compactions	$[1, 2^8]$	1
max_background_flushes	$[110]$	1
write_buffer_size	$[1, 15 * 10^7]$	2^{26}
max_write_buffer_number	$[1, 2^7]$	2
min_write_buffer_number_to_merge	$[1, 2^5]$	1
max_bytes_for_level_multiplier	$[5, 15]$	10
block_size	$[1, 5 * 10^5]$	2^{12}
level0_file_num_compaction_trigger	$[1, 2^8]$	2^2
level0_slowdown_writes_trigger	$[1, 2^{10}]$	0
level0_stop_writes_trigger	$[1, 2^{10}]$	36

4 Evaluation

4.1 Setup

We tuned ten RocksDB parameters to maximize IO operations throughput (IOPS). Critically for an embedded key-value like RocksDB to be efficient when performing its IO operations [14, 30]. Maximizing IOPS yields improvement to the system regardless of the application characteristic. Table 1 has descriptions and ranges of the parameters. We initialized our BO models with a single random observation to highlight our models' efficiency.

4.1.1 Experiment goals.

- Success criteria for the tuner is to **converge faster** and find the **most performant** IO throughput.
- Highlight the efficiency of cluster-based multi-task approach in exploiting system decomposability.

4.1.2 Benchmark. We used RocksDB's benchmark tool *db_bench* to model Facebook's social graph traffic per the parameters reported in [8]. It runs 50 million queries in fifteen-minutes. This workload has a mixture of all RocksDB operations: 78% GET, 13% PUT, 6% DELETE, and 3% Iterate, the workload patterns change every 5000 operation reflecting real workload. A detailed discussion of this benchmark is in Cao et al. [8], who also argues that this is more realistic benchmark than the popular YCSB [10]. We documented the benchmark command in the appendix. Every experiment repeated five times, with the variance visualized in the plots.

4.1.3 Libraries. We used GPyTorch (v1.3) [18] for the GP implementation. BoTorch (v0.3) [3] for the implementation of the *Expected improvement* [40] acquisition function in our implementation. We used Microsoft NNI v2.0 [29] implementations of TPE, Random, and GP(NNI) in the baseline.

4.1.4 Hardware. Google Cloud general compute instance e2-standard-4 (4 CPUs, 16GB RAM).

4.1.5 Baselines. Table 2 documents the baselines we used to maximize RocksDB's IOPS.

Table 2. Alternative surrogate models as baselines. The background has a short introduction to these methods 2.3.

Method	Use case
TPE [4]	Handles discrete parameters.
GP (NNI) [29]	Standard $O(n^3)$ implementation.
Random [5]	Low effective search dimensions.
Additive kernel [15]	Low-dimensions decomposability.
Default	RocksDB v6.17 default settings.
BoTorch [3]	Efficient GPyTorch $O(n^2)$ GP.

4.2 IOPS performance

We are interested in finding the best set of parameters that maximizes IOPS given a limited computational budget. We set a budget of 100 optimization steps to perform parameter search and reported the best-found IOPS in Figure 7.

The multi-task approach utilized the additional knowledge and found a configuration that other tuners could not find given the budget. The multi-task approach achieved a $x1.38 \pm 0.06$ increased throughput than RocksDB default settings. While Clustered multi-task achieved comparable results to other tuners, it consistently achieved them and achieved them fast. This is visible in the next subsection. The other baselines showed comparative optimization results, which leads us to investigate the next important metric, convergence speed.
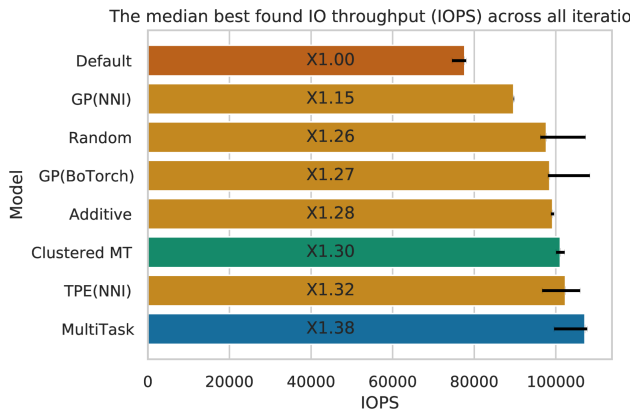

Figure 7. The best IO throughput found in 100 steps with the median of five runs reported with error bars showing the minimum and maximum achieved IOPS. The higher the IO throughput, the better. The middle text calculates the ratio of improvement over the default.

4.3 Convergence

An essential property of a tuner is to find the best configurations quickly. Computer systems evaluation is expensive to come by, and the computational budget of 100 we set is too generous. Here we evaluate each model convergence speed.

Figure 6 shows the rate of convergence. Both multi-task and clustered multi-task (*Clustered MT*) approaches can find better configurations and find them sooner than the baselines. It took *Clustered MT* 10 iterations consistently to find

configurations that provide $x1.3$ improvement over the default, and it took the next non-cluster-based surrogate model 60 iterations to reach comparable results. However, the non-clustered multi-task approach was able to find better configurations (x1.38) in the long run (sixty steps) but was unstable where some experiments had worse performance.

It is worth noting *GP(NNI)'s* flatline. The standard GP implementation started throwing memory exceptions and timeout due to its inability to handle the ten parameters. We show a more efficient implementation of GP in *GP(BoTorch)* which utilizes GPyTorch [18] as discussed in the background 2.3.

In summary, we showed that by providing the tuner with additional context of the system, it could model the search space more efficiently than other models.

5 Related work

Structured optimization. Both BOAT [12] and ProBO [27] demonstrated that adding structure to the surrogate model of the BO loop provides significant convergence speedup. BOAT allows the user to define semi-structured models: parametric models to capture trends and GP to generalize and aggregate.

This work distinguishes itself by providing a straightforward mechanism for injecting expert knowledge by exploiting correlated targets through MultiTask learning. We could not evaluate against them as ProBO is not publically available, while BOAT requires substantial work to integrate.

Multi-fidelity optimization. Multi-task learning has many similarities to the multi-fidelity literature [23]. Both approaches are identical for all intended purposes, as in a multi-fidelity approach with an exact number of observations for every objective, it will use an ICM kernel similar to the standard multi-task approach. The only key difference is the philosophy of optimization.

In the multi-fidelity framework, the low-fidelity targets are cheap to evaluate objectives that the user does not care about (in this paper, it would be the three adjacent objectives). The primary objective is modeled as the high-fidelity objective and a function of the lower fidelities. Since in this paper we actively chose the secondary objective, from the user's perspective, these secondary objectives are essential to optimize as well, which makes the problem better posed as multi-task learning rather than multi-fidelity.

Multi-objective optimization(MOO) [6] shares the philosophy of our work. In MOO, a Pareto frontier is calculated for all objectives, and the optimizer aims to balance the trade-off between optimizing the objectives according to the user's preference. Our work requires the user to identify dependent tasks, so the complex trade-off must be calculated as the user wants to maximize them all equally.

Reinforcement learning.(RL) [39] successfully applied for computer system optimization [25, 31, 34, 37]. It has the
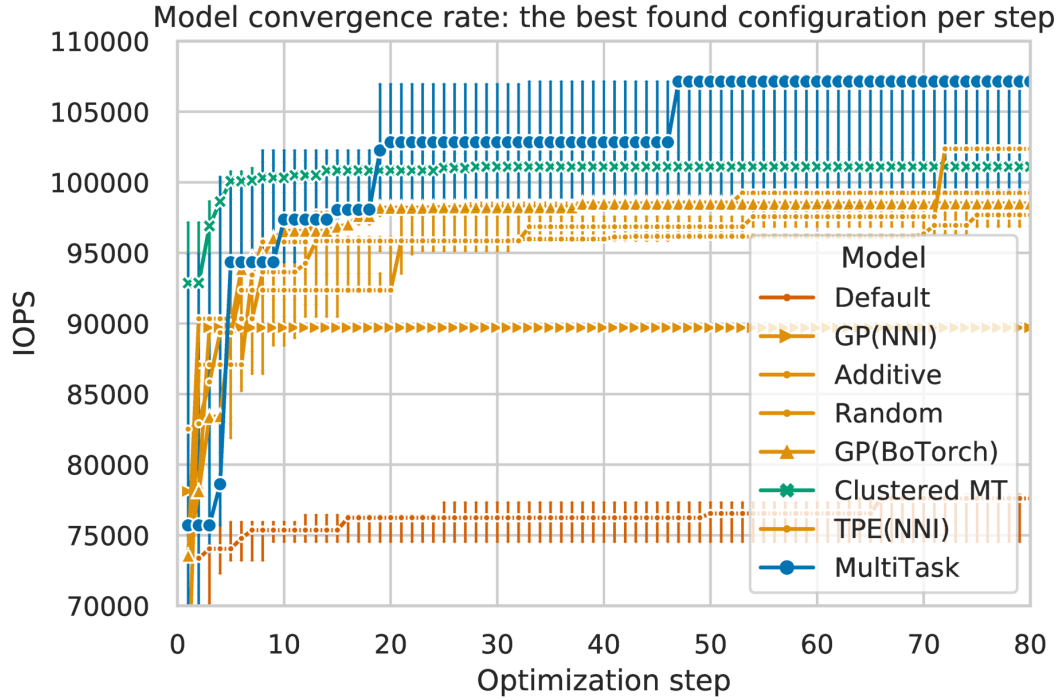
Figure 6. This plot reports the best configurations the tuner finds in every step. The X-axis shows the number of steps the optimizer took. The Y-axis represents the median best-found configuration. The error bar shows the best and worst run in five benchmark runs. The higher the IOPS, the better. The figure is cut to 80 steps since nothing changes after.

advantage of handling a large number of parameters efficiently and is extremely easy to use. However, it suffers from several drawbacks: first, it is susceptible to initializing seed due to the large number of stochastic processes used internally; this problem discussed in details in [19]; secondly, it requires many training samples; thirdly, its results are uninterpretable and hard to debug; and finally, it is challenging to include expert knowledge in the model.

Our approach protects against these issues: GP are resilient against the impact of a bad seed [32]. The multi-task structure provides an interpretation of the model performance to the system expert and easy to understand mechanism to inject their knowledge into the model. Finally, the multi-task structure provides each training step with many more samples, leading to faster convergence.

Self-tuning databases This work can fall under self-tuning databases literature. Schmied et al. [35] proposes a pipeline of machine learning techniques to tackle the large parameter space. The pipeline reduces the parameter space and selects the top ten promising parameters and their ranges to perform the BO or RL loop to optimize the main objective. Their work is orthogonal to our approach. The pipeline can be used as a preprocessing step to select more parameters to optimize and then utilize the multi-task approach to provide faster convergence.

RocksDB optimizations. Ouaknine et al. [30] tuned six RocksDB parameters manually to achieve x11 throughput. While Dong et al. [14] improved space and read efficiency by tuning different six parameters. Both cases show the need for an auto-optimizer for RocksDB to achieve its potential.

6 Conclusion

This work presents an efficient auto-tuner for RocksDB that handles its high-dimensional space. The tuner exploits alternative observable metrics and structural decomposability to converge faster and reduce the dimensional space. We utilize multi-task learning to provide an accessible mechanism for expressing structure in the model. The methodology is universal and can be applied to systems other than RocksDB as long as they expose their internal components metrics.

We demonstrated the tuner's effectiveness by tuning ten parameters to maximize IO throughput. The tuner outperformed the default configuration by x1.35 in 10 iterations, compared to the other state-of-the-art methods requiring 60 iterations.

Acknowledgments

References

[1] Mauricio A Alvarez, Lorenzo Rosasco, and Neil D Lawrence. 2012. Kernels for Vector-Valued Functions: A Review. *Foundations and Trends® in Machine Learning* 4, 3 (2012), 195–266.

[2] Jason Ansel and Cy Chan. 2010. PetaBricks. *XRDS: Crossroads, The ACM Magazine for Students* 17, 1 (2010), 32. https://doi.org/10.1145/1836543.1836554

[3] Maximilian Balandat, Brian Karrer, Daniel R Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. 2019. BoTorch: Programmable Bayesian Optimization in PyTorch. *arXiv preprint arXiv:1910.06403* (2019). arXiv:1910.06403

[4] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. *Advances in neural information processing systems* 24 (2011), 2546–2554.

[5] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *Journal of machine learning research* 13, Feb (2012), 281–305.

[6] Julian Blank and Kalyanmoy Deb. 2020. Pymoo: Multi-Objective Optimization in Python. *IEEE Access* 8 (2020), 89497–89509.

[7] Stephen Boyd, Lin Xiao, Almir Mutapcic, and Jacob Mattingley. 2007. Notes on Decomposition Methods. *Notes for EE364B, Stanford University* 635 (2007), 1–36.

[8] Zhichao Cao, Siying Dong, Sagar Vemuri, and David HC Du. 2020. Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook. In *18th {USENIX} Conference on File and Storage Technologies ({FAST} 20)*. 209–223.

[9] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. 2015. Apache Flink: Stream and Batch Processing in a Single Engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* 36, 4 (2015).

[10] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing*. 143–154.

[11] Valentin Dalibard. 2016. *A Framework to Build Bespoke Auto-Tuners with Structured Bayesian Optimisation*. Technical Report. Cambridge University, Department of Computer Science and Technology.

[12] Valentin Dalibard, Michael Schaarschmidt, and Eiko Yoneki. 2017. BOAT: Building Auto-Tuners with Structured Bayesian Optimization. In *Proceedings of the 26th International Conference on World Wide Web - WWW '17*. ACM Press, New York, New York, USA, 479–488. https://doi.org/10.1145/3038912.3052662

[13] Andy Dent. 2013. *Getting Started with LevelDB*. Packt Publishing Ltd.

[14] Siying Dong, Mark Callaghan, Leonidas Galanis, Dhruba Borthakur, Tony Savor, and Michael Strum. 2017. Optimizing Space Amplification in RocksDB.. In *CIDR*, Vol. 3. 3.

[15] David Duvenaud, Hannes Nickisch, and Carl Edward Rasmussen. 2011. Additive Gaussian Processes. *arXiv preprint arXiv:1112.4394* (2011). arXiv:1112.4394

[16] RocksDB Facebook. 2020. RocksDB Parameters. *GitHub* (Nov. 2020).

[17] RocksDB Facebook. 2021. RocksDB Homepage. Facebook.

[18] Jacob R. Gardner, Geoff Pleiss, David Bindel, Kilian Q. Weinberger, and Andrew Gordon Wilson. 2019. GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration. *arXiv:1809.11165 [cs, stat]* (Jan. 2019). arXiv:1809.11165 [cs, stat]

[19] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. 2017. Deep Reinforcement Learning That Matters. *arXiv preprint arXiv:1709.06560* (2017). arXiv:1709.06560

[20] James Hensman, Nicolo Fusi, and Neil D Lawrence. 2013. Gaussian Processes for Big Data. *arXiv preprint arXiv:1309.6835* (2013). arXiv:1309.6835

[21] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. In *International Conference on Learning and Intelligent Optimization*.

[22] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. 2017. Population Based Training of Neural Networks. (Nov. 2017). arXiv:1711.09846

[23] Kirthevasan Kandasamy, Gautam Dasarathy, Jeff Schneider, and Barnabás Póczos. 2017. Multi-Fidelity Bayesian Optimisation with Continuous Approximations. In *International Conference on Machine Learning*. PMLR, 1799–1808.

[24] Andy Liaw and Matthew Wiener. 2002. Classification and Regression by randomForest. *R news* 2, 3 (2002), 18–22.

[25] Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. 2017. Device Placement Optimization with Reinforcement Learning. *CoRR* abs/1706.0 (June 2017). arXiv:1706.04972

[26] Luigi Nardi, Bruno Bodin, Sajad Saeedi, Emanuele Vespa, Andrew J. Davison, and Paul H. J. Kelly. 2017. Algorithmic Performance-Accuracy Trade-off in 3D Vision Applications Using HyperMapper. *arXiv:1702.00505 [cs]* (March 2017). arXiv:1702.00505 [cs]

[27] Willie Neiswanger, Kirthevasan Kandasamy, Barnabas Poczos, Jeff Schneider, and Eric Xing. 2019. ProBO: Versatile Bayesian Optimization Using Any Probabilistic Programming Language. *arXiv preprint arXiv:1901.11515* (2019). arXiv:1901.11515

[28] Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. 1996. The Log-Structured Merge-Tree (LSM-Tree). *Acta Informatica* 33, 4 (1996), 351–385.

[29] OpenSource. 2020. Microsoft/Nni. Microsoft.

[30] Keren Ouaknine, Oran Agra, and Zvika Guz. 2017. Optimization of Rocksdb for Redis on Flash. In *Proceedings of the International Conference on Compute and Data Analysis*. 155–161.

[31] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd C Mowry, Matthew Perron, Ian Quah, Siddharth Santurkar, Anthony Tomasic, Skye Toor, Dana Van Aken, Ziqi Wang, Yingjun Wu, Ran Xian, and Tieying Zhang. 2017. Self-Driving Database Management Systems. In *CIDR'17*. https://doi.org/10.1109/JSTSP.2015.2425798

[32] Carl Edward Rasmussen. 2003. Gaussian Processes in Machine Learning. In *Summer School on Machine Learning*. Springer, 63–71.

[33] Documentations RocksDB. [n.d.]. Tuning RocksDB on Spinning Disks. https://github.com/facebook/rocksdb/wiki/Tuning-RocksDB-on-Spinning-Disks/b903ab5a834350c5ca439bde75bcbc928a849d9e.

[34] Michael Schaarschmidt, Alexander Kuhnle, Ben Ellis, Kai Fricke, Felix Gessert, and Eiko Yoneki. 2018. LIFT: Reinforcement Learning in Computer Systems by Learning From Demonstrations. (2018). arXiv:1808.07903

[35] Thomas Schmied, Diego Didona, Andreas Döring, Thomas Parnell, and Nikolas Ioannou. 2020. Towards a General Framework for ML-Based Self-Tuning Databases. *arXiv preprint arXiv:2011.07921* (2020). arXiv:2011.07921

[36] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando De Freitas. 2016. Taking the Human out of the Loop: A Review of Bayesian Optimization. *Proc. IEEE* (2016). https://doi.org/10.1109/JPROC.2015.2494218

[37] Ankur Sharma, Felix Martin Schuhknecht, and Jens Dittrich. 2018. The Case for Automatic Database Administration Using Deep Reinforcement Learning. (2018), 1–9. arXiv:1801.05643

[38] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In *Advances in Neural Information Processing Systems*. 2951–2959.

[39] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement Learning : An Introduction*. MIT Press, Cambridge Mass.

Springer, 507–523.

[40] Emmanuel Vazquez and Julien Bect. 2010. Convergence Properties of the Expected Improvement Algorithm with Fixed Mean and Covariance Functions. *Journal of Statistical Planning and inference* 140, 11 (2010), 3088–3095.

[41] Michel Verleysen and Damien François. 2005. The Curse of Dimensionality in Data Mining and Time Series Prediction. In *International*

Work-Conference on Artificial Neural Networks. Springer, 758–770.

[42] Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando de Feitas. 2016. Bayesian Optimization in a Billion Dimensions via Random Embeddings. *Journal of Artificial Intelligence Research* 55 (2016), 361–387.