Comparing Black-Box Optimization Methods for Online DBMS Tuning

Cong Ding, Gagan Gopinath, Rui Pan, Ziyi Zhang
Group 15

1 Introduction

Modern database management systems (DBMSs) expose hundreds of configuration knobs that collectively determine their runtime behavior. Those knobs control, for example, the amount of memory used for cache and the maximum number of concurrent reads. If the knobs are optimally configured, the database systems can achieve higher performance and efficiency. The traditional practice for manual DBMS tuning is where the database administrators (DBAs) use their expertise and manually experiment by changing individual knobs to understand their effectiveness on the entire workload. However, manual DBMS tuning is a daunting task even for expert DBAs because of the following reasons: **(1) Complexity:** Modern DBMSs have hundreds of interdependent configuration knobs, and most knobs take in continuous values. As a result, the configuration space for DBMS tuning is orders of magnitude larger than those of traditional systems (e.g., file systems) [8]. **(2) Hardware-dependent**: Different hardware might have different optimal configurations. e.g. a configuration that is good for SSD might not work for HDD [1]. **(3) Workload-dependent**: The optimal configuration for one application might be sub-optimal for another [19]. **(4) DBMS-dependent**: Two DBMSs may use different names for the same knob [19]. **(5) Inter-dependent**: The knobs are also dependent of each other (configuring one knob could impact the other knobs). Moreover, different DBMSs use different internal structures, so optimal configurations are not generic across different DBMSs. As a result, recent research has focused on the automatic tuning of the configuration knobs.

To automate this process, approaches like writing static rules (hard-coded) to configure the knobs [20] were introduced, but these approaches were not flexible. They could fail to work well on a new set of unseen workloads which were not thought about during the development of these rules. As they tend to be very rigid, it was complicated to alter the rules by adding new functionalities. Also, they fail to achieve improvement in some of the possible optimizations.

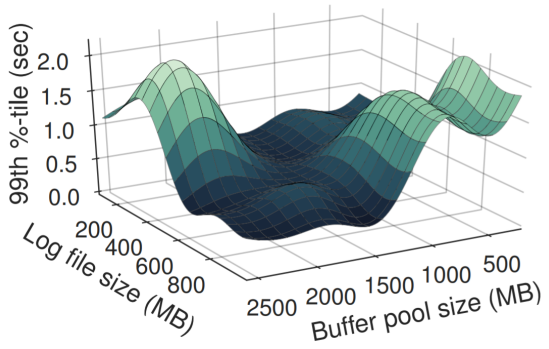

Figure 1: Performance measurements for the YCSB workload running on MySQL (v5.6) using different configuration settings [19].

Recently, there has been several automatic tuning methods and ML-based automatic tuning used to solve this problem, i.e., Bayesian Optimization, Reinforcement Learning, Deep Neural Networks, Gaussian Process Regression, etc. [2, 20, 22]. These approaches tend to perform better than the previous approaches taken as they can identify some of the inherent correlations and optimize the configurations accordingly to maximize the objective. Most of these approaches are also called black-box optimization methods, which means that the databases are treated as black-boxes and will give a performance metric for each configuration they are asked to run on. Therefore, it is important for the optimization methods to use the information gathered so far in each stage.

The most prominent ML-based approaches for database tuning are Bayesian Optimization and Gaussian Process [20]. However, the lack of comparison studies makes it difficult for users to decide which optimization method to employ given the production environment. We compare the efficacy of existing online black-box optimization methods for DBMS knob tuning given the importance to understand the use of specific methods for specific workloads on production-level DBMS systems and the lack of prior comparison studies in this area.

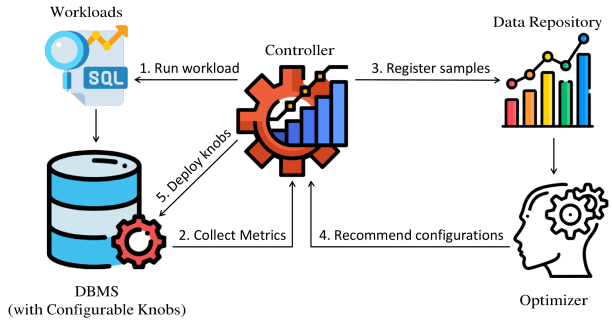

Figure 2: Workflow of online DBMS tuning.

2 Background

2.1 Online Database Tuning

State-of-the-art DBMS tuning services like Ottertune [19] mostly employ offline tuning, which requires the customers to collect enough configuration samples and build a knowledge base in advance. Then the optimizer could make use of the knowledge to featurize the current workload and leverage the corresponding tuning strategy. It can be expensive to build such a knowledge base. Besides, it's nearly impossible to iterate over all kinds of workloads. If the current workload cannot be identified, it will be hard for the optimizer to give a reasonable recommendation.

We evaluate the optimization approaches in an online fashion. The optimizer starts from an empty data repository and gathers the knowledge as the tuning proceeds. To explore more about the configuration space at the beginning, we run 10 random configurations for the first 10 iterations. Then the optimizer will leverage the data samples to give us recommendations and balances exploration and exploitation.

2.2 High-Dimensional Bayesian Optimization

Bayesian optimization (BO) is a well-known technique for database tuning problems. It has been employed by many state-of-the-art tuning services [19]. Here, we give a brief introduction to Bayesian Optimization.

BO is often used to optimize an unknown and expensive objective function. It can converge towards good function values and avoid evaluating bad configurations. BO leverages a surrogate model of the objective function and refines the model as more data points are evaluated. The acquisition function selects the next point to evaluate, while it balances *exploration* and *exploitation* by choosing candidate points from uncertain and promising regions.

While BO is an effective way for database tuning, it suffers from the curse of dimensionality. Due to the limitation of high-dimensional BO, Wang et al. proposed Random EMbedding Bayesian Optimization (REMBO) [21] to address the problem of scaling to high dimensions. The motivation of REMBO is that although the objective function has a high-dimension setting, it is often the case that the optimization problem has a low effective dimensionality. Kanellis et al. reported that tuning just 5 "important" knobs can achieve 99% of the performance obtained by tuning many knobs [8]. REMBO randomly generates a projection matrix to transform a low-dimensional vector to a high-dimensional one. We can view this way as the high-dimensional search space being "projected" to a low-dimensional one. Then a low-dimensional Bayesian optimization could be used to find a good solution in this low-dimensional space.

3 Design

We design our online DBMS tuning workflow as shown in Fig 2. The controller is responsible for running workloads and collecting metrics. After evaluating some configuration, we register the current setting and result in the data repository. The optimizer learns from previous samples and recommends a configuration to the controller. The controller then deploys the new configuration and starts the next iteration.

We choose three different black-box optimization strategies for database tuning.

Uniform random search. We implement the uniform random search algorithm (RANDOM) as our baseline. We specify a lower bound and upper bound for each knob. The algorithm randomly picks a value within the range from a uniform distribution. Then, we will evaluate the configuration and record the best throughput value we have seen.

Bayesian optimization. We employ the implementation of Bayesian Optimization in MLOS [5], which is an open-source parameter tuning library maintained by Microsoft. For the first 10 iterations of each tuning session, we randomly pick configurations to evaluate. After that, we evaluate the configuration proposed by the optimizer. However, we set the optimizer to generate a random point every ten iterations so that it can fully explore the configuration space and escape from the local optima.

REMBO. We introduce REMBO on top of our Bayesian Optimization implementation. The user should

specify the number of dimensions in the embedding space in advance. The algorithm first generates a random projection matrix, and optimizes upon the low-dimensional space. For each point proposed by the acquisition function, we project the point to the original space. Note that the point may be projected outside the boundaries, the algorithm will pick the nearest point on the boundary to avoid invalid configuration. Then, we evaluate the projected point and set the throughput as the target objection value of the corresponding low-dimensional embedding. Finally, the Bayesian optimizer will proceed to the next iteration.

4 Evaluation

4.1 Experiment Setup

We use Nautilus[1], a framework for streamlining the evaluation of DBMS configurations, to run experiments. Nautilus takes in different configurations, runs them on different workloads and hardware configurations, and returns the metrics to the users. In our experiments, we use throughput as a metric of performance. We use PostgreSQL [18] as an example production-level DBMS.

We conduct the tuning experiments on three workloads – YCSB-A, YCSB-B [4], and TPC-C [15]. YCSB is a common set of workloads for evaluating the performance of different "key-value" and "cloud" serving stores. YCSB-A is update-heavy (50% reads, 50% updates) and YCSB-B is read-heavy (95% reads, 5% updates). TPC-C is an online transaction processing (OLTP) benchmark. It is a mixture of read-only and update-intensive transactions that simulate the activities found in complex OLTP application environments. We run 100 iterations on each tuning session and 5 random seeds on each optimization strategy.

The hardware we run our experiments on is a single CloudLab [16] node on the Wisconsin $c220g5$ cluster. Each node is equipped with two Intel Xeon Silver 4114 10-core 2.20 GHz CPUs, 192GB of RAM, one 1 TB HDD, and one 480 GB SATA SSD [3].

4.2 RANDOM, BO, and REMBO

We first evaluate the above three black-box optimization approaches by tuning 27 preselected knobs and plotting the optimum throughput achieved over each iteration in Fig 3. Unless otherwise specified, we set the default number of embedding space dimensions for REMBO to be 4.

[1]https://github.com/uw-mad-dash/nautilus

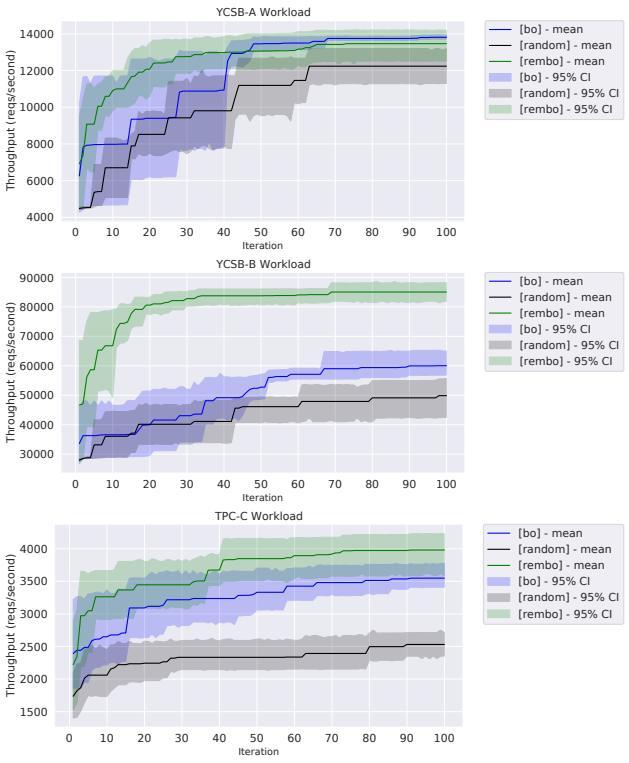

Figure 3: Optimum throughput achieved tuning 27 knobs over iterations on three workloads.

REMBO and BO constantly outperform Random Search across three workloads as we expected. REMBO performs better than BO on YCSB-B and TPC-C, while BO slightly outperforms REMBO in the later iterations on YCSB-A workload. Since the REMBO optimizer can only see a subspace, it is reasonable to assume that REMBO cannot fully explore the configuration space, thus BO is likely to outperform REMBO at the 100th iteration. However, 27 knobs may be too many for BO to handle. As a result, for YCSB-B and TPC-C, we notice that REMBO actually achieves a higher throughput. With more iterations, BO can explore more configuration space and converge at a higher throughput.

We also observe a stair-shaped pattern when we employ the BO or Random Search optimizer. When the optimizer configures the value of a few important knobs, the big performance leap will be captured. In contrast, REMBO uses a low-dimensional embedding for DBMS tuning. As it distributes the important knobs (dimensions) to the embedding space, we no longer observe such patterns in REMBO experiments. Instead, the throughput curves grow in a more continuous way.

Moreover, we notice that REMBO achieves approximately 90% of the best throughput of the current tuning session in the first 20 iterations. The surprisingly

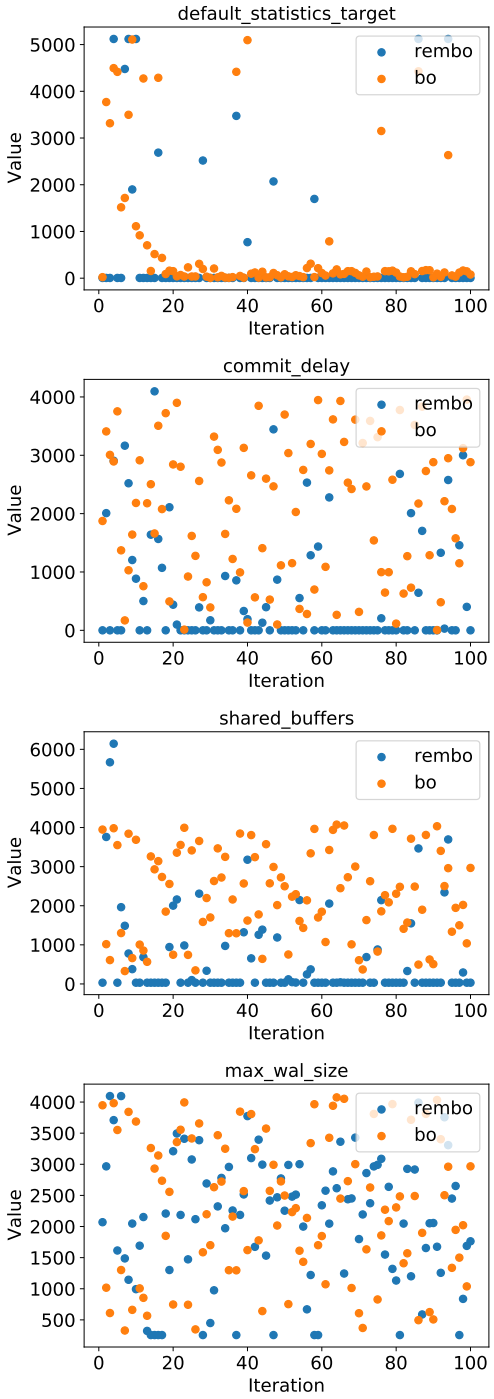

Figure 4: Values of four important knobs during a tuning session on workload YCSB-B.

fast convergence speed can be leveraged to accelerate DBMS tuning significantly. To figure out why REMBO converges fast, we inspect four important knobs and plot their values at each iteration in Fig 4. We can see that REMBO can only optimize upon a restricted subspace, so the values of each knob are more concentrated during iterations. As a result, REMBO can explore the space far more efficiently than BO does if the projection matrix is set appropriately. Whether REMBO generates the beneficial projection matrix is also one of the sources of the variance. In addition, the clipping which happens during projection would be another probable reason. When REMBO projects the embedding point to the original space, it is likely to fall out of the boundary. Since we specify the lower bound and upper bound of each knob in advance, we clip the value of each knob into the proper range to make sure it's a valid configuration. We find that the clipping happens on 22 knobs on average across all five random seeds and all three workloads. Clipping helps the optimizer choose the more "extreme" values compared to normal BO. For some important knobs (e.g., `default_statistics_target`), extreme values near lower or upper bounds may lead to a significant performance leap.

4.3 Tuning a Few Important Knobs for Near-Optimal Performance

Modern DBMSs have hundreds of interdependent configuration knobs and most knobs take in continuous values. As a result, the configuration space for DBMS tuning is orders of magnitude larger than those of traditional systems (e.g., file systems). Recent work [8] has found that with YCSB-A on Cassandra, tuning just five knobs can achieve 99% of the optimal performance achieved by tuning many knobs. This approach filters out the "important" knobs that have the most significant impact on the performance, thus reducing the configuration search space and accelerating database tuning. We conduct experiments by varying the number of knobs tuned and report our findings. In Section 4.2 where we present the canonical results, we tune 27 knobs, and in this section, we reduce the number of knobs to 9.

Impact on Throughput across Tuning Methods and Workloads. The performance degradation due to tuning fewer knobs vary across workloads and tuning methods. For example, for BO and RANDOM on the YCSB-B workload, we observed improvements in the final throughput instead of degradation. On the other hand, for BO on TPC-C, we observe a degradation of ∼20%. On average, across the three workloads, we observe a throughput loss of 95.2% on the optimal throughput achieved by the best tuning method. In general, our

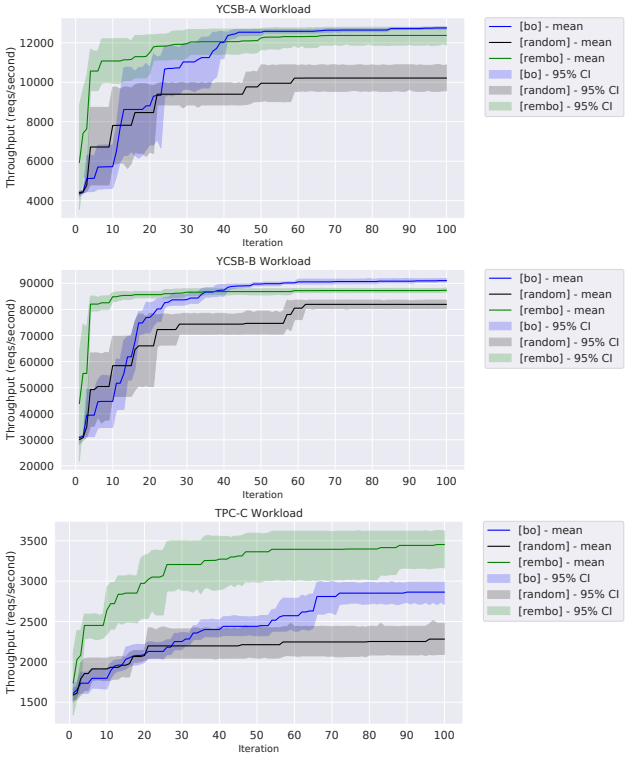

Figure 5: Optimum throughput achieved tuning 9 knobs over iterations on three workloads.

findings are in accordance with [8]. However, we note that although we used the same settings as those in [8], the performance degradation is inconsistent across workloads and tuning methods, and that the amount of degradation is higher than those reported in [8]. We attribute these discrepancies to the fact that the 9 most important knobs we hand-picked were not necessarily the optimal set of knobs that would result in a near-optimal final throughput.

BO's Performance Improvements in a Search Space with a Lower Dimension. The motivation for the REMBO algorithm aims at efficiently solving Bayesian optimization problems with high dimensions, in which the traditional Bayesian optimization techniques have trouble scaling to. With a fewer number of knobs being tuned, the configuration search space will have a less level of dimension, and our assumption is that compared to tuning a large number of knobs, BO will have performance wins over REMBO in a configuration space with a lower dimension. We speculate that this is because in the 27-dimensional search space, 100 iterations are not enough for BO (in which the whole space is searched) to find a better configuration than REMBO (in which only a low-dimensional subspace is searched).

Our experiment results in Section 4.2 show that REMBO achieves a throughput win of (0.97×, 1.42×, 1.14×) over BO for the three workloads (YCSB-A, YCSB-B, TPC-C). After reducing the number of knobs from 27 to 9, REMBO's throughput win drops to (0.97×, 0.95×, 1.23×) over the three workloads. On average, after reducing the number of knobs from 27 to 9, REMBO's relative performance win over BO suffers an 11% performance degradation, which matches our conjectures. However, we note that this general trend might not apply to all workloads. For example, for the TPC-C workload, after reducing the number of knobs, BO suffers from a reduction on the final throughput of around 27% while REMBO only suffers from minor performance degradation.

4.4 Different Dimensions of REMBO

The users need to specify the number of dimensions when employing REMBO. We conduct experiments by varying the number of REMBO dimensions to explore how it would affect the tuning performance.

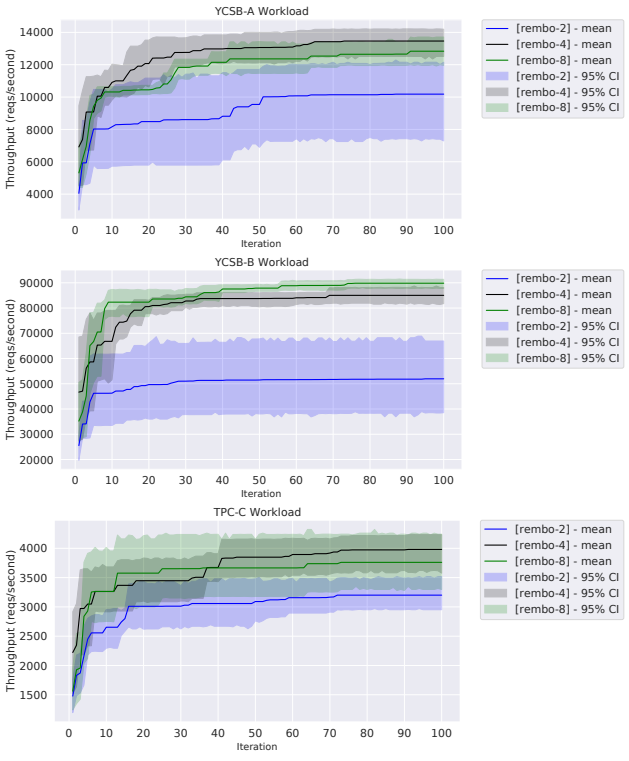

Figure 6: Optimum throughput achieved over iterations on three workloads.

REMBO is based on the intuition that only a few "effective" dimensions would have an impact on the objective functions. In the DBMS tuning context, we should set

the number of dimensions to be bigger than important knobs. From Fig 6, REMBO with 2-dimensional embeddings (REMBo-2) gives us the worst throughput and a relatively large variance since the number of dimensions cannot include all the important knobs.

For REMBO with 4-dimensional embeddings (REMBO-4) and 8-dimensional embeddings (REMBO-8), the performance gap is small. REMBO-4 outperforms REMBO-8 on YCSB-A and TPC-C, while REMBO-8 achieves a higher peak throughput on YCSB-B. We think 4 is a reasonable number to include all effective dimensions, and REMBO-8 suffers from a larger embedding space so that 100 iterations may not be sufficient. In this case, we don't want the number of dimensions to be smaller than the effective dimensions. A higher-dimensional space, however, will need more efforts to explore, although it has a bigger potential to give us better configuration.

5 Related Work

Automatic DBMS Configuration Tuning. Tuning configurations of modern DBMSs automatically is an active area in database studies [6, 10].

Database researcher employed rule-based and heuristic methods for configuration tuning to achieve high performance for particular DBMSs or some certain knobs. BestConfig [23] exploits limited resources to automatically find the optimal setting for a given workload.

Recently, machine learning methods have emerged as an effective approach for configuration tuning [11]. Ottertune [19] is the state-of-the-art configuration tuning system which leverages Bayesian Optimization and Gaussian Process to recommend knob configurations. CDBTune [22] aims at cloud databases and employs deep deterministic policy gradient (DDPG) [13] to find the optimal setting in high-dimensional configuration space.

Online Tuning Algorithms. Bayesian Optimization is a popular approach for many system configuration tuning problems [17]. It can explore the state space efficiently while the evaluation of the target function is expensive. RelM [9] firstly employs Bayesian Optimization to automatically tune the memory allocation for data analytic systems. iTuned [7] automates parameter tuning by finding high-impact parameters and high-performance parameter settings using. Ottertune [19] also uses Gaussian Process to fit the surrogate model. Alabed et al. [2] employs Bayesian optimization to find parameters that maximize RocksDB's I/O throughput.

Reinforcement learning is another type of approach that is leveraged in database tuning. CDBTune [22] takes use of DDPG for cloud database tuning. QTune [12] employs DS-DDPG for query-aware tuning.

6 Future work

Future directions of this project include the following:

- **Dive Deeper into our Existing Experiment Result.** For example, in Section 4.3, we notice that although we emulated the experiment setup in [8], we are seeing discrepancies in our results such as the degree of throughput loss and the generalization characteristic across workloads. We plan to investigate further on these findings and try to reason about these differences.
- **Combining REMBO and BO for the Best of Both Worlds.** In Section 4.2, we note that while REMBO has the fastest convergence rate, it does so by only optimizing upon a restricted subspace, making the values of each knob more concentrated. On the other hand, BO explores upon the whole configuration space. It would be interesting to try to combine REMBO and BO to leverage the fast convergence rate and the chance to search over the whole configuration space for the best of both worlds.
- **Evaluate Other HDBO Methods.** Aside from REMBO, there are a number of other approaches on high-dimensional BO (HDBO). For example, to utilize low-dimensional structures, Hashing-enhanced Subspace BO (HeSBO) [14] relies on hashing and sketching to reduce surrogate modeling and acquisition function optimization to a low-dimensional space. Evaluating these other methods comprehensively would provide more insights on database tuning.

7 Contributions Breakdown

Cong worked on running experiments on CloudLab, collecting results, and plotting the figures. For the writeup of the final report, all four of us worked on it together (Rui mainly worked on Section 4.3 & 6, Cong mainly worked on Section 4.2 & 4.4, Ziyi mainly worked on expanding Section 1 & 2.2, and Gagan mainly worked on Section 2.1 & discussion of Section 4).

References

[1] Documentations RocksDB. [n.d.]. Tuning RocksDB on Spinning Disks.

[2] S. Alabed and E. Yoneki. High-dimensional bayesian optimization with multi-task learning for rocksdb. In Proceedings of the 1st Workshop on Machine Learning and Systems, EuroMLSys '21, page 111–119, New York, NY, USA, 2021. Association for Computing Machinery.

[3] CloudLab. Cloudlab hardware.

[4] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC '10, page 143–154, New York, NY, USA, 2010. Association for Computing Machinery.

[5] C. Curino, N. Godwal, B. Kroth, S. Kuryata, G. Lapinski, S. Liu, S. Oks, O. Poppe, A. Smiechowski, E. Thayer, and et al. Mlos. Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning, Jun 2020.

[6] B. Dageville and M. Zait. Sql memory management in oracle9i. In VLDB'02: Proceedings of the 28th International Conference on Very Large Databases, pages 962–973. Elsevier, 2002.

[7] S. Duan, V. Thummala, and S. Babu. Tuning database configuration parameters with ituned. Proceedings of the VLDB Endowment, 2(1):1246–1257, 2009.

[8] K. Kanellis, R. Alagappan, and S. Venkataraman. Too Many Knobs to Tune? Towards Faster Database Tuning by Pre-Selecting Important Knobs. 2020.

[9] M. Kunjir and S. Babu. Black or white? how to develop an autotuner for memory-based analytics. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, pages 1667–1683, 2020.

[10] E. Kwan, S. Lightstone, A. Storm, and L. Wu. Automatic configuration for ibm db2 universal database. Proc. of IBM Perf Technical Report, 2002.

[11] G. Li, X. Zhou, and L. Cao. Machine learning for databases. Proc. VLDB Endow, 14(12):3190–3193, 2021.

[12] G. Li, X. Zhou, S. Li, and B. Gao. Qtune: A query-aware database tuning system with deep reinforcement learning. Proceedings of the VLDB Endowment, 12(12):2118–2130, 2019.

[13] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, 2015.

[14] A. Nayebi, A. Munteanu, and M. Poloczek. A framework for Bayesian optimization in embedded subspaces. In Proceedings of the 36th International Conference on Machine Learning, volume 97 of Proceedings of Machine Learning Research, pages 4752–4761. PMLR, 09–15 Jun 2019.

[15] F. Raab. Tpc-c - the standard benchmark for online transaction processing (oltp). In The Benchmark Handbook, 1993.

[16] R. Ricci, E. Eide, and The CloudLab Team. Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications. USENIX ;login:, 39(6), Dec. 2014.

[17] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. Advances in neural information processing systems, 25, 2012.

[18] M. Stonebraker and L. A. Rowe. The design of postgres. In Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data, SIGMOD '86, page 340–355, New York, NY, USA, 1986. Association for Computing Machinery.

[19] D. Van Aken, A. Pavlo, G. J. Gordon, and B. Zhang. Automatic database management system tuning through large-scale machine learning. In Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD '17, page 1009–1024, New York, NY, USA, 2017. Association for Computing Machinery.

[20] D. Van Aken, D. Yang, S. Brillard, A. Fiorino, B. Zhang, C. Bilien, and A. Pavlo. An inquiry into machine learning-based automatic configuration tuning services on real-world database management systems. Proc. VLDB Endow., 14(7):1241–1253, Mar. 2021.

[21] Z. Wang, F. Hutter, M. Zoghi, D. Matheson, and N. de Freitas. Bayesian optimization in a billion dimensions via random embeddings, 2016.

[22] J. Zhang, Y. Liu, K. Zhou, G. Li, Z. Xiao, B. Cheng, J. Xing, Y. Wang, T. Cheng, L. Liu, M. Ran, and Z. Li. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In Proceedings of the 2019 International Conference on Management of Data, SIGMOD '19, page

415–432, New York, NY, USA, 2019. Association for Computing Machinery.

[23] Y. Zhu, J. Liu, M. Guo, Y. Bao, W. Ma, Z. Liu, K. Song, and Y. Yang. Bestconfig: tapping the performance potential of systems via automatic configuration tuning. In Proceedings of the 2017 Symposium on Cloud Computing, pages 338–350, 2017.