Performance Roulette: How Cloud Weather Affects ML-Based System Optimization

Johannes Freischuetz
University of Wisconsin – Madison
Madison, WI 53706
freischuetz@cs.wisc.edu

Konstantinos Kanellis
University of Wisconsin – Madison
Madison, WI 53706
kkanellis@cs.wisc.edu

Brian Kroth
Microsoft Gray Systems Lab
Madison, WI 53703
bpkroth@microsoft.com

Shivaram Venkataraman
University of Wisconsin – Madison
Madison, WI 53706
shivaram@cs.wisc.edu

Abstract

As system complexity, workload diversity, and cloud computing adoption continue to grow, both operators and developers are turning to machine learning (ML) based approaches for optimizing systems.ML based approaches typically perform measurements to evaluate candidate system configurations to discover the most optimal configuration. However, it is widely recognized that cloud systems can be effected by "cloud weather", i.e., shifts in performance due to hardware heterogeneity, interference from co-located workloads, virtualization overheads, etc. Given these two trends, in this work we ask: how much can performance variability during training affect ML approaches applied to systems?

Using DBMS knob configuration tuning as a case study, we present two measurement studies that show how ML based optimizers can be affected by noise. This leads to four main observable problems: (1) there exist of very sensitive configurations, the performance of which do not transfer across machines of the same type, (2) unstable configurations during training significantly impact configuration transferability, (3) tuning in an environment with non-representative noise degrades final performance in the deployment environment, (4) sampling noise causes a convergence slowdown. Finally, we propose a set of methods to mitigate the challenges in measurements for training ML based system components.

1 Introduction

The application of machine learning (ML) to problems in systems has been a hot topic for the last decade because of significant performance improvements over traditional methods [25, 29, 28, 5, 26, 27, 18, 34, 15]. A common aspect in many of these ML-based approaches is the need for representative, accurate, and comparable performance data of a target components, such that an ML model can perform meaningful optimization [4, 15, 18, 29, 26, 27, 25, 34, 7, 32, 20].

However, a number of prior measurement studies have shown that various levels of the system stack have high levels of performance variability [23, 2, 24]. These problems are exacerbated from several sources on shared infrastructure. Common reasons for performance variation on such infrastructure include virtualization overheads and noisy neighbors [21, 22]. Other studies have also shown performance variance between identical virtual machine (VM) configurations [9, 30].

Most existing approaches to mitigate noisy performance measurements involve taking many samples to infer the mean and coefficient of variation (CoV). This can be problematic in practice for two reasons. Firstly, Maricq et. al. [19] found that even modest increases in hardware CoV (above 4%), increase the requirement to hundreds of measurements. Second, we find that the degree of variability

Machine Learning for Systems Workshop at 37th NeurIPS Conference, 2023, New Orleans, LA, USA.

depends on the configuration and workload used. This makes it prohibitively expensive to profile the performance of a large system [16], let alone collect enough data for training an ML model [12].

A combination of requiring representative data measurements for ML, and the inherent noise in shared infrastructure environments raises the following questions: How much does performance variability during tuning impact the *transferability* (i.e., the ability for a configuration to be run on another host with similar performance) of learned configurations across machines? How much does this same performance variability affect the convergence of ML methods used to optimize systems?

To answer the above questions we perform two case studies tuning PostgreSQL knob configurations using SMAC [13], a popular Bayesian Optimizer (BO). We use clusters of systems from Cloudlab [8] and Azure to highlight the differences between the two infrastructures. Our key findings include:

- Noise from inherent hardware variability or collocated processes disproportionately affect a significant portion configurations leading to a new phenomenon of *non-transferable configurations* having up to, 60.8% throughput degradation when transferred to near identical bare-metal hardware.
- The existence of *unstable* configurations (i.e., configurations which perform inconsistently even on a single machine), significantly degrade the transferability of the best learned configuration.
- Performing tuning in the presence of noise is necessary to achieve good performance in noisy environments. For example, for a deployment with 25% background CPU noise, training in a similar setup can lead to 18% improvement over training in a noise-free environment.
- However, performance tuning in the presence of noise slows the rate of convergence by up to 64%.

Based on our findings, we propose new directions for systems and ML researchers to mitigate the effect of noisy performance data. While some ML methods that can handle noisy data [10, 17, 11], these require more samples, and/or oracle information about the variance of a sample. We instead propose techniques to modify sampling to improve data quality or use previously unused metrics.

2 Case Studies

In this section, we perform measurement based case studies to understand how noise, that arises from variability in hardware and shared computing environments [22], can affect ML methods. In particular, we focus on understanding two aspects of ML-based auto-tuning. First, we study how well configurations found using ML transfer across machines, which is crucial for cases where training is first performed in an offline setting in order to avoid negative impact to production workloads, a frequent concern. Following that, we study how noise can affect the convergence of ML-based optimizers both in terms of the number of iterations and the quality of the best configuration found.

2.1 Case Study 1: Poor Configuration Transfer

We first study how ML-based autotuners are impacted by hardware and cloud performance variability in the context of configuration transferability. For our experiments we tune DBMS configurations and use a state-of-the-art Bayesian Optimization (BO) based optimizer, SMAC [13]. In the context of DBMS tuning, SMAC has been shown to outperform alternatives [35], especially when the configuration search space is high-dimensional and heterogeneous (i.e., consists of both numerical and categorical knobs). For all experiments we tune PostgreSQL using SMAC for 100 iterations with 10 identical but random initialization points (similar to prior works [35, 15]).

2.1.1 Transferability of Learned Configurations on Bare-metal Hardware

To study how implicit variance in hardware performance can affect the ML optimizer, we run 109 tuning trials across 60 isolated bare metal machines on CloudlLab [8]. This results in 109 *best-performing configurations*, as discovered by SMAC. We then measure the performance of each such configuration, by running them across 15 different Cloudlab nodes. Finally, we compute the standard deviation across all 15 runs, to test how transferable best-performing configurations are.

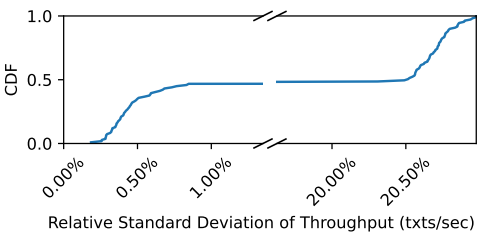

Figure 1: Empirical CDF of the DBMS relative performance standard deviation for the best-performing configurations found during the 109 tuning trials on 15 nodes.

Figure 1 shows the empirical CDF of relative standard deviation from every best-performing configuration. Surprisingly, we find that *more than half (i.e.,* 54%*)*

of these configurations have high variance and are non-transferable. This high variance is caused by the fact that the "best" learned configurations perform worse on 4 out of 15 machines, degrading on average by 39.6% and up to 60.8% when used on a new machine. Examining system performance metrics, such as disk throughput and CPU performance via micro-benchmarks, revealed (1) no obvious correlations and (2) that the new machines were performing within the small bounds expected of hardware variance. Thus, we hypothesize that there exist *non-transferable* configurations where even the presence of minor variances can significantly impact performance.

Finally, it is worth noting that in this set of 109 best-performing configurations, the median throughput of all transferable configurations is *only* 4.4% worse than the median of all non-transferable configurations found, implying stability does not incur a significant performance cost. The performance improvement for non-transferable configurations may be the result of the auto-tuning model overfitting to some aspect of the specific machine instance.

2.1.2 Tuning in the Cloud

To study the impacts of tuning the cloud, we ran 50 trials, evenly distributed across 10 `Standard_D8d_v5` [14] Azure VMs: a general purpose, non-bursting VM with non-bursting ephemeral disks (i.e., local SSDs). The choice of this particular VMs, was to remove bursting and remote disks as sources of performance variance seen in some other VM configuration offerings. We also ran micro-benchmarks on all the VMs and found no significant difference in their variance, or average performance. The question we wish to answer here is: are modern tuning processes resilient to sample performance variance, and in turn always able to generate a usable configuration?

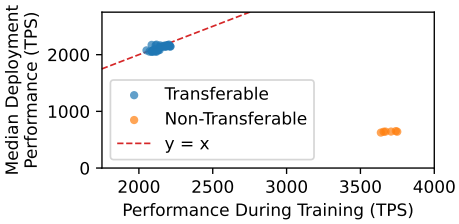

Figure 2: Clustering of stable and unstable configurations when transferring best learned configurations to all nodes in the cluster.

The first thing that we notice from this experiment is 4 of the 10 initial configurations show a very large performance variance across VMs, as well as *within the same VM* across trials. The performance seen was very bimodal once degrading *nearly* 95%; e.g., one configuration got approximately 2030 TPS on some trials and 132 TPS on other trials, on the same VM. Similar results are seen in in all 4 of the initial unstable configurations. We term these configurations *unstable*, as they are distinct from non-transferable configurations because of the inherent cloud-originated intra-VM variance.

Interestingly, we find that on tuning trials where the these unstable configurations perform well, they perform significantly better than other configurations. This influences the optimizer in such that when it perform well on an unstable configuration during bootstrapping, the final configuration generated is also non-transferable. Figure 2 shows this effect clearly showing that any best-performing configuration that performed far above the median during training, degraded significantly during deployment. Thus, the existence of unstable configurations in the cloud, and their impact on tuning, motivates the need for methods to *collect* representative samples, and *detect* anomalous configurations.

2.1.3 Using Configurations Autotuned with Noise.

Next, we consider how noise simulated with collocated processes present during tuning can affect the final performance of the best-performing configuration in settings with different levels of demand. We use `stress-ng` [1] restricted to a certain CPU utilization level as our collocated process. In particular, this experiment is useful to understand if using a noisy cloud VM with noisy neighbors for tuning, but deploying the final configuration in a noise-free machine is tenable (or vice-versa).

		Train		
		Noise Free	25% CPU	50% CPU
Deploy	Noise Free	$\mathbf{3467 \pm 5.0\%}$	$3285 \pm 13.7\%$	$3237 \pm 13.8\%$
	25% CPU	$2316 \pm 16.0\%$	$\mathbf{2699 \pm 5.4\%}$	$2288 \pm 15.5\%$
	50% CPU	$1769 \pm 11.2\%$	$1772 \pm 11.2\%$	$\mathbf{2018 \pm 7.1\%}$

Table 1: Throughput (txts/sec) achieved (on *deployment* system) by the best configuration found by the optimizer during a tuning trial (performed on *train* system).

In Table 1, we see that with a state-of-the-art optimizer (i.e., SMAC), training in an environment with noisy neighbors representative of the deployment environment leads to the best performance (i.e., the diagonal entries in the table). This is because having resource contention in the system during

tuning leads the tuner to find configurations that under/over utilize the available system resources during deployment. Given that it is hard to control for similar levels of contention during training and deployment, especially in dynamic cloud computing environments, these results highlight the need for new techniques to address this gap.

2.2 Case Study 2: Effects of Sampling Noise on Auto-Tuning

Finally, we investigate how DBMS auto-tuner convergence is affected when we inject artificial Gaussian noise during sampling. Convergence is an important area of study for DBMS auto-tuning because of the high cost of measuring a single sample, often taking at least 15 minutes [33, 15]. We use the same experimental tuning setup as Section 2.1.1, using 10 bare-metal nodes on CloudLab, evenly distributed across 5 levels of noise. We also ensure that all the nodes perform similarly on non-transferable configurations to make the results more comparable.

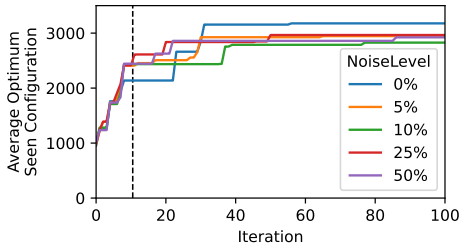

Figure 3: Convergence curves for DBMS auto tuning with injected artificial Gaussian noise. Configurations to the left of the dotted line are identical bootstrap configurations.

To inject noise, as before, we run the optimizer suggested configuration on PostgreSQL measuring its performance. Then, we emulate noise by *degrading* the measured performance (i.e., throughput) using a Gaussian prior. In particular, if P represents the measured performance, σ represents our chosen relative standard deviation, then $P^* = P \times (1 - |\Delta|)$, where $\Delta \sim \mathcal{N}(0, \sigma^2)$. Finally, we inform the optimizer that the performance of the suggested configuration is P^*. Here, we considered five levels of noise in $[0\%, 50\%]$.

The results of this experiment are shown in Figure 3. We observe that noise degrades convergence of the underlying optimizer (i.e., SMAC). We can quantify this in terms of time-to-accuracy, the rate at which it takes the baseline optimizer to reach the same performance of a noisy sample optimizer. For the noisy samples with 5%, 10%, 25%, and 50% relative standard deviation, we see convergence slowdown of 52.3%, 61.0%, 38.0%, 64.0% from the baseline respectively. While these samples are not strictly decreasing, this can be explained by our small sample size, and an outlier run in the 25% noise level. The convergence slowdown is because many configurations which were high performers are degraded significantly, which not only wastes the sample, but also influences the acquisition function in the optimizer. As dynamic noise is an inherent issue in cloud-native environments, this further motivates the need for methods to address noise during ML-based optimization.

3 Discussion and Future Work

Previously, we observed that the types of noise we studied can mislead an ML-based optimizer into finding sub-optimal configurations. This is because existing ML-based optimizers lack additional context regarding the execution environment, which prevents them from adapting to a multitude of constraints that exist in complex cloud deployments. These findings motivate the need for designing a *noise-aware* tuning system that can handle multiple objectives: to identify optimal configurations that are resilient to noise, while also performing well (enough) in its absence.

We propose three research directions, which we envision can be used independently, or combined, to build such a system in the future. Firstly, one direct approach is to evaluate configurations across a set of nodes to ensure stability across the set. While this can seamlessly be parallelized, it increases the cost of tuning significantly. Secondly, we plan to study if we can measure the amount of noise that the system is experiencing using side-channel techniques [31, 6]. This can be used to derive an accurate prediction of noise-free performance and fed to the optimizer as an additional input. Finally, we also plan to study if we can calculate relative performance compared to a baseline by running the candidate configurations collocated with a baseline configuration [3]. This will help mitigate the impacts of noise as both instances will be subject to the same noise.

4 Conclusion

In this paper we studied the significance of performance variance when tuning database systems. We analyzed the noise on these systems from the perspective of hardware variance, as well as the impact of noisy neighbors. We found that both can lead to significant impact on the tuning process. To mitigate these issues, we proposed new directions that can lead to the discovery of noise resilient configurations in the future.

References

[1] stress-ng. `https://wiki.ubuntu.com/Kernel/Reference/stress-ng`. Accessed: 2023-02-02.

[2] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai. The impact of performance asymmetry in emerging multicore architectures. In *32nd International Symposium on Computer Architecture (ISCA'05)*, pages 506–517, 2005.

[3] L. Bulej, V. Horkỳ, P. Tuma, F. Farquet, and A. Prokopec. Duet benchmarking: Improving measurement accuracy in the cloud. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, pages 100–107, 2020.

[4] S. Cereda, S. Valladares, P. Cremonesi, and S. Doni. Cgptuner: a contextual gaussian process bandit approach for the automatic tuning of it configurations under varying workload conditions. *Proceedings of the VLDB Endowment*, 14(8):1401–1413, 2021.

[5] S. Chaudhuri and V. Narasayya. Self-tuning database systems: A decade of progress. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, page 3–14. VLDB Endowment, 2007.

[6] C. Disselkoen, D. Kohlbrenner, L. Porter, and D. Tullsen. Prime+ abort: A timer-free high-precision l3 cache attack using intel {TSX}. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 51–67, 2017.

[7] M. Dorier, R. Egele, P. Balaprakash, J. Koo, S. Madireddy, S. Ramesh, A. D. Malony, and R. Ross. HPC storage service autotuning using variational- autoencoder -guided asynchronous bayesian optimization. In *2022 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, sep 2022.

[8] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Ricart, L. Landweber, C. Elliott, M. Zink, E. Cecchet, S. Kar, and P. Mishra. The design and operation of CloudLab. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 1–14, Renton, WA, July 2019. USENIX Association.

[9] B. Farley, A. Juels, V. Varadarajan, T. Ristenpart, K. D. Bowers, and M. M. Swift. More for your money: Exploiting performance heterogeneity in public clouds. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, New York, NY, USA, 2012. Association for Computing Machinery.

[10] P. I. Frazier. A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

[11] P. I. Frazier, W. B. Powell, and S. Dayanik. A knowledge-gradient policy for sequential information collection. *SIAM Journal on Control and Optimization*, 47(5):2410–2439, 2008.

[12] T. Hoefler and R. Belli. Scientific benchmarking of parallel computing systems: Twelve ways to tell the masses when reporting performance results. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, New York, NY, USA, 2015. Association for Computing Machinery.

[13] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In C. A. C. Coello, editor, *Learning and Intelligent Optimization*, pages 507–523, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[14] A. Jia, M. McKittrick, S. Tsinaroglou, and J. Pelley. Ddv5 and ddsv5-series. `https://learn.microsoft.com/en-us/azure/virtual-machines/ddv5-ddsv5-series`, oct 2022. Accessed: 2023-10-06.

[15] K. Kanellis, C. Ding, B. Kroth, A. Müller, C. Curino, and S. Venkataraman. Llamatune: Sample-efficient dbms configuration tuning. *Proc. VLDB Endow.*, 15(11):2953–2965, sep 2022.

[16] T. Leesatapornwongsa, C. A. Stuardo, R. O. Suminto, H. Ke, J. F. Lukman, and H. S. Gunawi. Scalability bugs: When 100-node testing is not enough. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, HotOS '17, page 24–29, New York, NY, USA, 2017. Association for Computing Machinery.

[17] B. Letham, B. Karrer, G. Ottoni, and E. Bakshy. Constrained bayesian optimization with noisy experiments, 2018.

[18] G. Li, X. Zhou, S. Li, and B. Gao. Qtune: A query-aware database tuning system with deep reinforcement learning. *Proc. VLDB Endow.*, 12(12):2118–2130, aug 2019.

[19] A. Maricq, D. Duplyakin, I. Jimenez, C. Maltzahn, R. Stutsman, and R. Ricci. Taming performance variability. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, OSDI'18, page 409–425, USA, 2018. USENIX Association.

[20] D. B. Prats, F. A. Portella, C. H. A. Costa, and J. L. Berral. You only run once: Spark auto-tuning from a single run. *IEEE Transactions on Network and Service Management*, 17(4):2039–2051, 2020.

[21] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, C. Pu, and Y. Cao. Who is your neighbor: Net i/o performance interference in virtualized clouds. *IEEE Transactions on Services Computing*, 6(3):314–329, 2013.

[22] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *Proc. VLDB Endow.*, 3(1–2):460–471, sep 2010.

[23] P. Sinha, A. Guliani, R. Jain, B. Tran, M. D. Sinclair, and S. Venkataraman. Not all gpus are created equal: Characterizing variability in large-scale, accelerator-rich systems. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '22. IEEE Press, 2022.

[24] D. Skinner and W. Kramer. Understanding the causes of performance variability in hpc workloads. In *IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005.*, pages 137–149, 2005.

[25] Z. Song, D. S. Berger, K. Li, and W. Lloyd. Learning relaxed belady for content distribution network caching. In *17th USENIX Symposium on Networked Systems Design and Implementation*, 2020.

[26] 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.

[27] 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, apr 2021.

[28] G. Wang, J. Xu, and B. He. A novel method for tuning configuration parameters of spark based on machine learning. In *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 586–593, 2016.

[29] H. Wang, S. Rafatirad, and H. Homayoun. A+ tuning: Architecture+application auto-tuning for in-memory data-processing frameworks. In *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 163–166, 2019.

[30] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, B. Smith, and R. H. Katz. Selecting the best vm across multiple public clouds: A data-driven performance modeling approach. In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, page 452–465, New York, NY, USA, 2017. Association for Computing Machinery.

[31] Y. Yarom and K. Falkner. Flush+ reload: A high resolution, low noise, l3 cache side-channel attack. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 719–732, 2014.

[32] Z. Yu, Z. Bei, and X. Qian. Datasize-aware high dimensional configurations auto-tuning of in-memory cluster computing. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 564–577, 2018.

[33] B. Zhang, D. Van Aken, J. Wang, T. Dai, S. Jiang, J. Lao, S. Sheng, A. Pavlo, and G. J. Gordon. A demonstration of the ottertune automatic database management system tuning service. *Proc. VLDB Endow.*, 11(12):1910–1913, aug 2018.

[34] 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.

[35] X. Zhang, Z. Chang, Y. Li, H. Wu, J. Tan, F. Li, and B. Cui. Facilitating database tuning with hyper-parameter optimization: A comprehensive experimental evaluation. *Proc. VLDB Endow.*, 15(9):1808–1821, jul 2022.