Wayfinder: Towards Automatically Deriving Optimal OS Configurations

Alexander Jung
Lancaster University
Lancaster, UK

Hugo Lefeuvre
The University of Manchester
Manchester, UK

Charalampos Rotsos
Lancaster University
Lancaster, UK

Pierre Olivier
The University of Manchester
Manchester, UK

Daniel Oñoro-Rubio
NEC Laboratories Europe GmbH
Heidelberg, Germany

Mathias Niepert
NEC Laboratories Europe GmbH
Heidelberg, Germany

Felipe Huici
NEC Laboratories Europe GmbH
Heidelberg, Germany

Abstract

Tuning operating systems configuration in order to obtain the maximum application performance is a hard problem. This is due to the extremely large size of the configuration space offered by modern OSes, and to the fact that it is generally explored manually. To address that issue, we propose to bring automation to the OS configuration space exploration process, in order to derive effortlessly and as quickly as possible optimal OS configurations for a given use case.

We present Wayfinder, a generic OS performance evaluation platform. Wayfinder is fully automated and ensures both the accuracy and reproducibility of results, all the while speeding up how fast tests are run on a system. Wayfinder is easily extensible and offers convenient APIs to (1) implement custom configuration space exploration techniques, (2) add new benchmarks and (3) support additional OS projects. We demonstrate Wayfinder's capacity to automatically and efficiently explore a LibOS' networking configuration space; as well as its ability to efficiently isolate parallel experiments to avoid noisy neighbors.

ACM Reference Format:
Alexander Jung, Hugo Lefeuvre, Charalampos Rotsos, Pierre Olivier, Daniel Oñoro-Rubio, Mathias Niepert, and Felipe Huici. 2021. Wayfinder: Towards Automatically Deriving Optimal OS Configurations. In *12th ACM SIGOPS Asia-Pacific Workshop on Systems (APSys'21), August 24–25, 2021, Hong Kong, China.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3476886.3477506

APSys'21, August 24–25, 2021, Hong Kong, China

ACM ISBN 978-1-4503-8698-2/21/08.
https://doi.org/10.1145/3476886.3477506

1 Introduction

Tuning operating systems via configuration options is fundamental to squeezing the best performance out of applications, whether in terms of throughput, delay or memory consumption, to name a few metrics. However, achieving such performance remains a bit of a dark art: countless articles and blogs regularly appear explaining how to performance-tune the underlying OS such that a particular application (*e.g.*, NGINX, Redis, MySQL, *etc.* [3, 22, 23]) can perform at its best. Such writings are the result of endless hours of trial-and-error and (often manual) measurements to understand which OS and application configuration options matter most, and what their values should be. This is particularly true as the popularity of the LibOS [5, 30] increases: a model advocating for a deep specialization of the kernel for a particular application or use case.

This status quo does not only mean that a select few can tackle the difficult issue of attaining the best application performance, but that even these experts may miss optimization opportunities because the configuration option exploration space is extremely large. In this paper we ask two fundamental questions:

- *Is it possible to automatically and intelligently derive optimal OS configurations for particular applications?* Ideally we would like users to simply select a target application and then let the system run a battery of experiments to sort and discover, hopefully in the shortest time possible, an optimal set of configuration options.
- *Is it possible to automatically gain insights into which options matter most?* Through trial-and-error experimentation and common sense we know that certain configuration options really matter in terms of performance (*e.g.*, disabling access logging in NGINX or enabling batching in the network stack), but would it be possible for an automated system to return a list of such options, or combination of options, such that we could learn about what influences performance?

Towards exploring these high level goals we introduce Wayfinder, a system which can automatically and efficiently explore a relatively large set of configuration options, and for each, measure the performance of the specific target application. While the ultimate target is general-purpose OSes such as Linux, our current prototype focuses on LibOSes because (1) their configuration option space is of small/medium size and (2) the build times are considerably smaller, thus allowing us to explore a larger space in a smaller amount of time. Wayfinder is at its core an automated benchmarking infrastructure. It supports several OSes including OSv [14], HermiTux [24], Unikraft [15] and Lupine [17], and adding support for more is effortless. The configuration search space for a given application can easily be described in configuration files and the system supports speeding up experiments by running them in parallel while enforcing the required isolation for each run to avoid noisy neighbor effects.

An important question in Wayfinder is: *given a large configuration space how should it choose which configuration options to explore for each performance experiment?* A naive and non-scalable approach would be to perform a grid search, successively trying all possible parameter configurations. Since exploring all configurations is too costly for all but a small number of applications, one typically has to limit the number of evaluations. Given such a fixed budget on the number of configurations, a random parameter search would likely miss good configurations. Due to the similarity of the problem to tuning (hyper-)parameters for complex machine learning models, Bayesian optimization and related methods developed for this purpose might achieve a good trade-off between finding a high performance configuration and doing so in a relatively short amount of time. Ultimately however, we envision using machine learning algorithms, and in particular neural network (NN) models. Recent advances in NN show promising results in reinforcement learning[21, 33], explainable ML[27, 28, 31], generative models[2, 8, 12, 13], and transfer learning[6, 7, 26]. We can exploit their capabilities in fitting complex distributions and their ability to transfer the learned knowledge to related problems allowing us to predict the best configurations for multiple applications and hardware setups. Furthermore, we can take advantage of explainable ML methods to highlight the importance of input features, and to detect possible input configurations that cause bottlenecks.

Towards this high level goal, in this early work we make the following research contributions:

- We introduce Wayfinder, a novel, generic and modular benchmark platform capable to automatically set configuration options, build operating systems and the target application, and test the latter's performance while recording system performance statistics (*e.g.*, memory consumption, CPU counters, *etc.*).
- A modular sub-system that allows for plugging different algorithms for deciding which configuration options to explore at each turn; in our prototype we evaluate this using grid, random and Bayesian optimization algorithms.
- A use case based on NGINX using Wayfinder to automatically find a performance optimum (in terms of requests per second), and to graphically show which parameters mattered the most in order to achieve such performance (*i.e.*, insights).

Wayfinder, as well as all configuration files used in this paper, are open-source and can be found at:

https://github.com/lancs-net/wayfinder.

2 Design Principles and Implementation

Wayfinder's main goal is to let users automatically and efficiently explore the wide design space offered by modern OSes. As such, we envision it to be beneficial in a range of use cases including searching for performance optimums, identifying configuration parameters having a strong impact on performance, and in general helping OS users and their designers to fairly compare several OS models over several metrics and in various scenarios. With this in mind, the design goals of Wayfinder are:

(1) To provide reproducible benchmarking environments for LibOS experimentation and to allow for full automation of the performance measurement process.
(2) To provide automated methods to efficiently explore the vast design space offered by modern LibOSes.
(3) To efficiently leverage the parallelism inherent in multicore hosts to speed up the tests while avoiding noisy neighbor effects arising from co-locating sensitive performance measurements on compute units sharing hardware resources (*e.g.*, caches).
(4) A modular architecture which enables the extensibility of various components, in particular 1) new user-defined exploration strategies; 2) new benchmarks; and 3) new LibOSes.

At a high level, Wayfinder addresses these requirements by 1) automating the entire evaluation process and uniquely describing each experiment through configuration files, thus making them reproducible; 2) offering 3 automated design space exploration strategies (grid and random search, Bayesian optimization); 3) allowing users to specify the required degree of isolation for each test and automatically inferring the possible parallelism between tests; and 4) offering convenient APIs so users can customize the parameterization process and extend Wayfinder's support for new exploration strategies, new benchmarks and OSes.

We begin the description of Wayfinder's architecture by noting that many OSes configuration parameters are set statically at build time – in particular in LibOSes which is

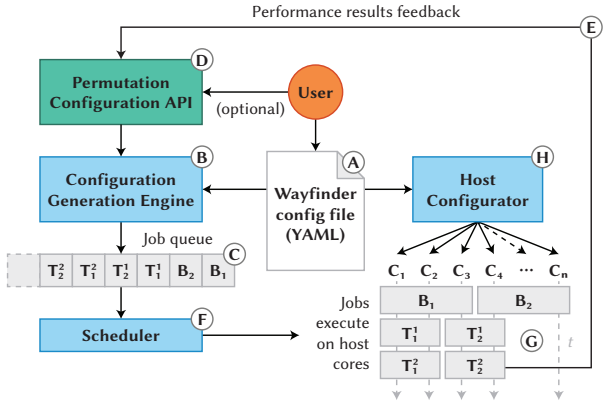

Figure 1: Overview of Wayfinder's architecture. An id i corresponds to a fixed set of values for all parameters, B_i corresponds to the build job for i, and T_i^j corresponds to the test job of iteration number j for i. C_1-C_N represent the host cores.

the model we target for our proof-of concept. As a result, an experiment run using Wayfinder will not only include *test jobs* but also *build jobs* (see Figure 1).

The user describes the experimental environment in a configuration file – point Ⓐ in the figure. Among other things, the file contains the list of OS configuration parameters to be varied and how they should be varied during the experiment. This is fed to the configuration generation engine Ⓑ, which produces a queue Ⓒ of OS build jobs and experiment run jobs according to the configuration file.

The generator can simply iterate over the entire exploration space by varying each parameter step by step, but can also be tuned by the user through a special API Ⓓ which uses feedback from the performance measurement results Ⓔ in order to intelligently search for performance optima in the exploration space without having to enumerate all possible parameter combinations.

Finally, a scheduler Ⓕ dispatches and executes jobs on the host cores Ⓖ based on job requirements in terms of physical resources and isolation *levels* (more on levels later). Various host global parameters, for example power management governors or the enabling/disabling of technologies such as Hyper-Threading or Turbo Boost, can be set according to user preferences Ⓗ.

In the rest of this section we give more details about some of Wayfinder's key features and describe how they address our design principles.

Experiment Setup: Configuration File. Experiments are provided to Wayfinder in YAML format. The YAML file includes the path to two important scripts: the *build script* that will be used to run build jobs, in other words, the configuring and building of the OS; and the *test script*, used to run the OS and measure performance metrics. The YAML configuration file also contains the list of OS configuration parameters which should be varied during the experiment as well as how they should be varied (e.g., on/off switches or ranges of values with a variation step). Finally, the configuration file includes various runtime fields, including: the number of iterations for each executed test; the number of cores required for the build and test jobs; timeouts for a particular run or whether the experiment should abort/stop; isolation requirements for test jobs; and, host-level configuration parameters (e.g., Hyper-Threading or CPU core pinning).

Build Jobs Parameterization. A central part of an experiment in Wayfinder is the list of OS configuration parameters to vary and how they should be varied. For example, the following YAML configuration snippet describes the variation of Unikraft's [35] TCP/IP stack (lwIP [4]) sender buffer size from 1MB to 100MB by increments of 1MB:

```
1  params:
2    - name: LWIP_SND_BUF # TCP sender buffer size
3      min:  1048576      # vary from 1MB
4      max:  104857600    # to 100MB
5      step: 1048576      # by steps of 1MB
6    - name: LWIP_RCV_BUF # Next parameter
7    # ...
```

This information is passed to the configuration generation engine, which will produce combinations of fixed values for each parameter. Each such combination will trigger the generation of a build job and one or several test jobs, according to the number of iterations chosen by the user. The build job executes the build script indicated in the YAML file after having generated a build configuration file such as `config.h` or set particular environment variables according to the OS model considered. The test jobs are generated and run similarly. For instance, in the example given above, Wayfinder will set `LWIP_SND_BUF` with the value of this parameter for the current iteration (starting at 1048576) before calling the build script which will read that value and construct the OS accordingly.

We assume that the build scripts include version checks for the sources of both the OS and the benchmark code (*e.g.*, force a `git checkout` before the build). Given that, an experiment in Wayfinder can be uniquely described by the YAML configuration file and the build/test scripts: it is thus reproducible.

Configuration Generation Engine and Exploration Strategies. The default behavior of the configuration generation engine is to naively generate a job for each possible combination of parameters. However, this does not scale well to the very large configuration space offered by modern OSes [16, 17]. To improve scalability, the configuration generation engine provides a customization API which allows users to implement custom parameter exploration logic and generate new

configuration parameters combinations based on several inputs, including, in particular, past parameters combinations and their corresponding performance results. These are fed back to the generation engine by Wayfinder (recall step Ⓔ in the architecture diagram). This opens up the possibility of intelligently searching for performance optima or identifying the parameters having the largest impact on performance using user-defined techniques or heuristics.

We implemented on top of the API three of such design space exploration strategies: *grid search*, going over the entire search space incrementally by setting each parameter's value from a predefined grid; *random search*, iterating over the space by setting parameters randomly; and *Bayesian optimization*, using Bayes' theorem to converge quickly on an optimal configuration point.

Parallelism vs. Performance Isolation. Given the large exploration space, on modern machines Wayfinder is faced with the conflicting goals of 1) leveraging parallelism on multiple cores to run as many jobs as possible simultaneously in order to minimize the experiment run time; and, 2) avoiding the noisy neighbor effects which occur when co-locating jobs on compute units sharing resources, for example Hyper Threads from the same core or cores sharing caches. While some jobs such as memory-intensive benchmarks would negatively affect each others when co-located, others such as build jobs or memory usage measurements would not.

To maximize parallelism while maintaining performance isolation for the jobs which require it, the user can declare, for each test job, the *level of isolation* required for its execution. We define 5 levels of isolation:

(1) **No isolation**: jobs in the level can be co-located, *i.e.* run on Hyper Threads of the same core;
(2) **Core**: two jobs in this level will not run on two Hyper Threads of the same core;
(3) **Cache**: jobs can not run on cores sharing a cache;
(4) **Socket**: jobs can not be co-located on the same socket;
(5) **Full isolation**: no other job can run in parallel.

Wayfinder automatically analyzes the hardware configuration of the host to determine its topology. The scheduler uses this information, along with job requirements to make informed decisions regarding test and build job placement Ⓖ. Build jobs are automatically set to the "no isolation" level. In addition, on recent CPUs, Wayfinder takes care of tuning the performance isolation vs. parallelism trade-off in a fine-grained fashion by leveraging Intel's Cache Allocation Technology [9] and Memory Bandwidth Allocator [10] mechanisms.

Further, in order to achieve OS-level isolation and consistency between jobs, Wayfinder leverages Linux's namespaces and control groups on the host. A privileged container environment, initialized using libcontainer [25], is used to build the OS and run the experiment with the correct environment variables values and build configuration files. This approach enables a fine level of resource control, including CPUs, RAM, network bandwidth, *etc.* Note that jobs are always pinned to compute units (Hyper Threads or CPU cores) and we do not schedule more than jobs in a Hyper Thread per core.

Table 1: New benchmarks/OSes porting effort.

Experiment	LibOS	Lines of Code			
		YAML	Build	test	Patch
Boot time & memory footprint	HermitTux	2	0	10	2
	Lupine	2	16	23	0
	OSv	2	9	33	9
	Rumprun	2	15	13	8
	Unikraft	2	10	74	7
TCP/IP	Unikraft	34	8	8	0

Adding New Benchmarks and OSes. Adding new benchmarks or OSes to Wayfinder is straightforward. Introducing a new benchmark is as simple as writing 1) a script to build an OS for the new benchmark (*build script*); and 2) a script to execute the OS in question (*run script*).

Porting a new OS to Wayfinder mostly consists of updating the configuration generation engine by specifying whether the build is configured through environment variables or generated headers, such as `config.h`.

As we focus on LibOSes, Wayfinder includes built-in metric monitors that are critical in this OS model, *i.e.*, boot time and memory footprint. These require slight updates to the LibOS and to the hypervisor if the OS in question is virtualized. The LibOS update is negligible and only involves adding a single line of inline assembly code at the end of the boot process to mark a timestamp helping in measuring the boot time. For virtualized LibOSes, the hypervisor/toolstack also needs to be slightly modified for precise boot time measurement, but once again this represents a very small effort. Note that Wayfinder ships with patches to include these modifications to several popular hypervisors (QEMU/KVM, Firecracker [1], `ukvm`/Solo5 [32, 36], and uHyve [18, 24]) and LibOSes (HermiTux, Unikraft, OSv, Rumprun, and Lupine Linux). As an example, adding HermiTux to Wayfinder took less than one hour of work. Table 1 reports the amount of LoC written to port new benchmarks (boot time, memory usage, and NGINX performance measurement) and to port several LibOSes (patch).

Implementation. Wayfinder consists of about 2K lines of Golang code. The current version includes numerous configuration files: build and test scripts for the aforementioned OSes, files for several benchmarks including networking performance measurements with NGINX, and files to define boot time and memory usage measurements.

3 Evaluation

Wayfinder allows to explore automatically and efficiently a potentially large configuration space. To evaluate this, we leverage our system to perform a comprehensive performance exploration of network configuration options in Unikraft [35] running the NGINX web server. We then show how Wayfinder can be used to efficiently find the best configuration in a minimum amount of steps. Finally, we study Wayfinder's capacity to isolate benchmarks running in parallel in order to avoid disturbances.

We ran all experiments on a server (2xIntel Xeon E5-2640) with Linux 4.19.0 and configured for maximum performance (isolcpus on core 0-1, HT and ASLR disabled, performance CPU governor). Unless otherwise stated, build jobs run with no isolation and experiment jobs run with full isolation.

TCP/IP Configuration Space Exploration. In order to demonstrate the ability of Wayfinder to automate configuration space exploration, we use a web server LibOS based on Unikraft and its NGINX port. The resulting project exposes 275 configuration options across multiple libraries, including the network stack, the scheduler and the NGINX application. As the exploration space is prohibitively large and several parameters are numeric, we focus our analysis on a small set of key network and application parameters. Specifically, from the lwIP library we vary the memory management model (malloc'ed — `MEMP_MEM_MALLOC` – vs. pool-based - `MEMP_MEM_MALLOC`), the virtio driver mode (poll vs. interrupt), and the size of the statically-allocated memory for TCP listeners (`LWIP_NUM_TCPLISTENERS`) as well as connected sockets (`LWIP_NUM_TCPCON`) from 4 to 64 by increments of 2^N. In parallel, we explore the performance impact of the access logging, caching and `Keep-Alive` options in NGINX.

Based on these parameters, Wayfinder automatically generates 512 unique LibOS images, reflecting all possible configuration permutations. Using the `wrk` HTTP performance tool, we generate a steady stream of HTTP requests for a static page of 612 bytes and measure the average server response rate. In each experiment we configure `wrk` to use 30 persistent HTTP connections, each running on a separate thread.

Our results (see Figure 2) demonstrate the significant impact that LibOS configuration options can have on performance: different configurations can result in a large range of performance results, from 239 reqs/sec (`L,K,H,T=64,N=64`) to 103470 reqs/sec (`K,P,T=32,N=32,O,C`) – a three orders of magnitude increase – even though we modify, in these two cases, the same application configuration options. From the results, we also highlight the low performance of configurations than enable logging and the positive impact of a high number of pre-allocated TCP buffers and HTTP Keep-Alive on performance.

In addition, we note that Unikraft's default configuration, shown by a red line in the figure, yields rather poor performance, only 33% of the throughput of the best configuration. This highlights the fact that defaults tend to be sub-optimal, and that a framework such as Wayfinder can help practitioners find performance optima much more quickly.

Design Space Exploration Strategies. Even if the configuration space of the previous experiment is relatively small (512 configurations), it takes nine hours to complete. Wayfinder implements exploration strategies whose goal is to find in a vast configuration space an optimum in as few steps as possible.

We evaluated the capacity of our 3 exploration strategies (grid and random search, Bayesian optimization) to find an optimal configuration in the Unikraft Nginx search space presented earlier (see Figure 2). For each strategy we vary the number of configurations that can be tested before ending the search and reporting the tested configuration giving the best performance. This number is set to be 10, 40, and 100.

Figure 3 reports the maximum NGINX throughput found as an average and stdev of 10 runs of each strategy for each number of iterations. For grid search, the starting point is randomized for each run. Even with a large number of iterations, the naive grid search rarely finds a configuration close to the maximum throughput possible. The relatively good performance of the random search are due to the small size of the design space: given a larger space, random search is unlikely to yield good results when the number of iterations is but a small fraction of the entire space. Unsurprisingly, the Bayesian Optimization gives the best performance, consistently finding the best configuration (or a close contender) even with small numbers of iterations (40 being 8% of the 512 configurations constituting the design space). This demonstrates the capacity of Wayfinder to quickly find an optimum in the vast configuration space offered by today's OSes.

Experimental Isolation The ability of Wayfinder to parallelize experiments introduces a trade-off between measurement precision and experiment run times, which can be controlled via a job isolation policy. To analyze the impact of parallelization, we use the best NGINX/Unikraft configuration (`K,P,T=32,N=32,O,C`) and explore the impact of different NUMA/core scheduling policy. Each experiment requires 3 dedicated cores (VM, QEMU VMM, `wrk`) and our server allows up to 6 parallel experiment executions. We explore four job isolation policies; pinning all instances on cores on the same NUMA node and pinning one instance on a core on a separate NUMA node from the other two instances. Flexible isolation policies can reduce resource fragmentation and increase experiment parallelization, but can decrease measured performance.

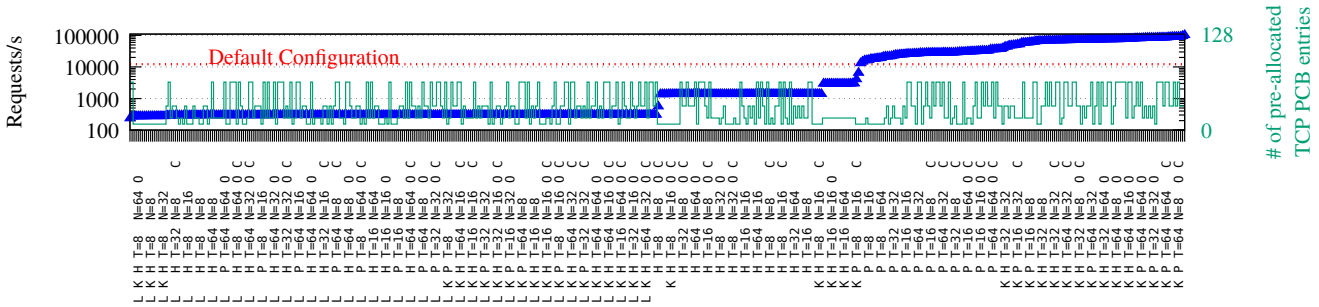

Figure 2: Performance of an NGINX/Unikraft LibOS for different configuration options (log scale). NGINX parameters: HTTP `Keep-Alive` (K), Caching (C), Access logging (L). lwIP parameters: Number of pre-allocated TCP PCB entries (T), Pool-based memory management (P), Heap-based memory allocation (H), Poll-based virtio driver (O).

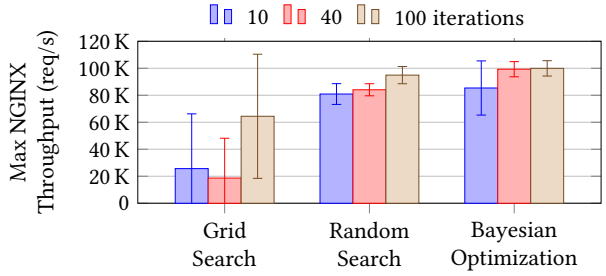

Figure 3: Maximum NGINX throughput found (average/stdev of 10 runs) using various exploration strategies with different iteration numbers.

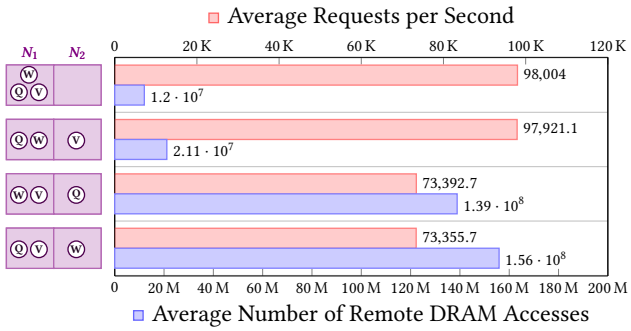

Figure 4: Average performance and total remote memory access count of six NGINX/Unikraft LibOS instances for different job isolation policies. N_n represents the physical socket, NUMA node and the placement strategy for the QEMU VMM (Q), `wrk` (W) and the VM (V).

Figure 4 presents the rate (req/sec) and total remote memory accesses (Intel PMU counter) measured across six parallel experiments, using the different job isolation policies. Our experiments show that maintaining NUMA node affinity between instances yields the highest average performance. Experimenters can achieve relatively high performance results with a flexible isolation policy, as long as the VMM and the `wrk` instances are on the same NUMA node. Scheduling them on separate sockets results in a 20% reduction in performance, due to the increased number of remote memory accesses. Network buffer allocation and NIC offloadings are implemented by the QEMU VMM, thus maintaining memory locality can greatly reduce access latency. It is worth highlighting that these observations are workload-specific and experimenters must perform an initial analysis of the precision and parallelization trade-offs.

4 Related Work

Benchmarking frameworks such as Wayfinder aim to automate the performance evaluation process not only to speed it up, but also to reduce human error and ensure that results are accurate and reproducible. In the domain of systems software, Auto-Pilot [37] is a generic framework easing and automating the process of running tests, measurements, and analysis tools. Shivam et al. [29] describe a similar set of automation techniques for server benchmarking, introducing, in particular, a component that takes as input the measurements from past experiments to decide how to define future tests. Pilot [20] is a framework for systems software benchmarking whose notable features include real-time result analysis and auto-detection of warm-up/tear-down phases, among others. All these projects expect to run within a POSIX-like environment, something that is not supported by all OSes. For example, many LibOSes [24] are not POSIX-compatible. Existing frameworks require significant effort to enable support for new OSes, including configuration/build process integration, execution environment setup, and implementation of

monitoring for critical metrics (*i.e.*, boot time, memory usage); this justifies our choice of starting from scratch with Wayfinder.

Several research efforts have explored the topic of automated configuration analysis. Tartler et al. [34] developed the first parameter analysis study of the Linux kernel, and identified several compile-time parameter configurations that result in invalid builds. LearnConf [19] is a static code analysis framework for JAVA applications that can predict performance-critical configuration parameters. Violet [11], is a configuration space analysis tool for large software projects using symbolic execution to construct all possible execution paths for a given set of configuration parameters. The system uses function latency estimations, measured using execution tracing, to predict the performance of each path. ATR [11] is an optimization framework for Redis deployments that uses ensemble learning to predict an optimal configuration. Wayfinder provides a novel approach to the problem of configuration space exploration, allowing to explore a wider configuration space by incorporating in experiment parameters across system layers and allowing precise performance estimation.

5 Conclusion

Tuning OS parameters to obtain optimal performance is a difficult problem. This stems from the wide configuration space of modern OSes, and the fact that it is generally explored manually. We introduced Wayfinder, a novel platform for automatic testing of OS performance across a set of different projects. Wayfinder streamlines the design space exploration process in order to quickly and intelligently derive optimal OS configurations for given use cases. It ships with support for several OSes, applications, and exploration strategies. It is also easily extensible to support more of each.

Wayfinder opens the door for the application of more advanced machine learning methods to the problem of finding optimal system configurations. The problem of optimizing parameters for OSes could benefit from novel machine learning methods that have shown promising performance in predicting and modeling the behavior of complex systems. Moreover, recent developments in transfer learning could be adopted to predict the performance of optimal configurations even for unseen systems and new applications. Wayfinder can contribute to this emerging research area in three ways: (1) providing an environment for automatically and efficiently collecting data, (2) establishing a standard method for computing metrics, and (3) performance measures for ML models applied to the problem of optimizing OS configurations.

6 Acknowledgements

This work has been partially funded by EU H2020 grant agreements 825377 (UNICORE), 871793 (ACCORDION) and the Next Generation Converged Digital Infrastructure (NG-CDI) Prosperity Partnership project funded by UK's EPSRC and British Telecom PLC under grant EP/R004935/1.

References

[1] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. 2020. Firecracker: Lightweight Virtualization for Serverless Applications. In *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, 419–434.

[2] Martin Arjovsky, Soumith Chintala, and L'eon Bottou. 2017. Wasserstein Generative Adversarial Networks. In *International Conference on Machine Learning*.

[3] Abhishek Dubey. 2019. Redis Best Practices and Performance Tuning. https://bit.ly/3hMqMvu.

[4] Adam Dunkels. 2001. Design and Implementation of the lwIP TCP/IP Stack. *Swedish Institute of Computer Science* 2 (2001), 77.

[5] Dawson R Engler, M Frans Kaashoek, and James O'Toole Jr. 1995. Exokernel: An operating system architecture for application-level resource management. *ACM SIGOPS Operating Systems Review* 29, 5 (1995), 251–266.

[6] Yaroslav Ganin and Victor Lempitsky. 2015. Unsupervised Domain Adaptation by Backpropagation. In *International Conference on International Conference on Machine Learning*.

[7] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *Conference on Computer Vision and Pattern Recognition*.

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*.

[9] A. Herdrich, E. Verplanke, P. Autee, R. Illikkal, C. Gianos, R. Singhal, and R. Iyer. 2016. Cache QoS: From concept to reality in the Intel® Xeon® processor E5-2600 v3 product family. In *Proceedings of the 22nd IEEE International Symposium on High Performance Computer Architecture*. 657–668. https://doi.org/10.1109/HPCA.2016.7446102

[10] Herdrich, Andrew J. and Cornu, Marcel David and Abbasi, Khawar Munir. 2019. *Introduction to Memory Bandwidth Allocation*. https://software.intel.com/content/www/us/en/develop/articles/introduction-to-memory-bandwidth-allocation.html

[11] Yigong Hu, Gongqi Huang, and Peng Huang. 2020. Automated Reasoning and Detection of Specious Configuration in Large Systems with Symbolic Execution. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI'20)*. USENIX Association, 719–734. https://www.usenix.org/conference/osdi20/presentation/hu

[12] Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical Reparametrization with Gumbel-Softmax. In *Proceedings International Conference on Learning Representations*.

[13] Diederik P. Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations*.

[14] Avi Kivity, Dor Laor, Glauber Costa, Pekka Enberg, Nadav Har'El, Don Marti, and Vlad Zolotarov. 2014. OSv: Optimizing the Operating System for Virtual Machines. In *Proceedings of the 22nd USENIX Annual Technical Conference (USENIX ATC'14)*. USENIX, 61–72. http://dl.acm.org/citation.cfm?id=2643634.2643642

[15] Simon Kuenzer, Vlad-Andrei Bădoiu, Hugo Lefeuvre, Sharan Santhanam, Alexander Jung, Gaulthier Gain, Cyril Soldani, Costin Lupu, Ştefan Teodorescu, Costi Răducanu, Cristian Banu, Laurent Mathy,

Răzvan Deaconescu, Costin Raiciu, and Felipe Huici. 2021. Unikraft: Fast, Specialized Unikernels the Easy Way. In *Proceedings of the 16th European Conference on Computer Systems* (Online Event, United Kingdom) *(EuroSys '21)*. Association for Computing Machinery, New York, NY, USA, 376–394. https://doi.org/10.1145/3447786.3456248

[16] Simon Kuenzer, Sharan Santhanam, Yuri Volchkov, Florian Schmidt, Felipe Huici, Joel Nider, Mike Rapoport, and Costin Lupu. 2019. Unleashing the power of unikernels with unikraft. In *Proceedings of the 12th ACM International Conference on Systems and Storage*. ACM, 195–195.

[17] Hsuan-Chi Kuo, Dan Williams, Ricardo Koller, and Sibin Mohan. 2020. A Linux in Unikernel Clothing. In *Proceedings of the 15th European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, 15 pages. https://doi.org/10.1145/3342195.3387526

[18] Stefan Lankes, Simon Pickartz, and Jens Breitbart. 2016. HermitCore: A Unikernel for Extreme Scale Computing. In *Proceedings of the 6th ACM International Workshop on Runtime and Operating Systems for Supercomputers (ROSS'16)*. ACM. https://doi.org/10.1145/2931088.2931093

[19] Chi Li, Shu Wang, Henry Hoffmann, and Shan Lu. 2020. Statically Inferring Performance Properties of Software Configurations. In *Proceedings of the 15th European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, Article 10, 16 pages. https://doi.org/10.1145/3342195.3387520

[20] Yan Li, Yash Gupta, Ethan L Miller, and Darrell DE Long. 2016. Pilot: A framework that understands how to do performance benchmarks the right way. In *2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 169–178.

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. In *NIPS Deep Learning Workshop*.

[22] MYSQL. 2021. MySQL Performance Tuning and Optimization Resources. https://www.mysql.com/why-mysql/performance/.

[23] Rick Nelson. 2014. Tuning NGINX for Performance. https://www.nginx.com/blog/tuning-nginx/.

[24] Pierre Olivier, Daniel Chiba, Stefan Lankes, Changwoo Min, and Binoy Ravindran. 2019. A Binary-Compatible Unikernel. In *Proceedings of the 15th ACM SIGPLAN/SIGOPSInternational Conference on Virtual Execution Environments (VEE) (VEE'19)*. ACM, 59–73.

[25] Open Container Initiative. 2021. *The runC CLI tool for spawning and running containers*. https://github.com/opencontainers/runc

[26] Sinno Jialin Pan and Qiang Yang. 2010. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering* 22 (2010), 1345–1359.

[27] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *International Conference on Knowledge Discovery and Data Mining*.

[28] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2017. Grad-CAM: Visual Explanations From Deep Networks via Gradient-Based Localization. In *International Conference on Computer Vision*.

[29] Piyush Shivam, Varun Marupadi, Jeffrey S Chase, Thileepan Subramaniam, and Shivnath Babu. 2008. Cutting Corners: Workbench Automation for Server Benchmarking.. In *USENIX Annual Technical Conference*. 241–254.

[30] Mark Silberstein. 2017. Leave your OS at home: the rise of library operating systems. https://www.sigarch.org/leave-your-os-at-home-the-rise-of-library-operating-systems/.

[31] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2013. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *CoRR* (2013).

[32] Solo5. 2021. *The Solo5 sandboxed execution environment for unikernels*. https://github.com/Solo5/solo5

[33] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (second ed.). The MIT Press. http://incompleteideas.net/book/the-book-2nd.html

[34] Reinhard Tartler, Daniel Lohmann, Julio Sincero, and Wolfgang Schroder-Preikschat. 2011. Feature Consistency in Compile-Time-Configurable System Software: Facing the Linux 10,000 Feature Problem. In *Proceedings of the 6th Conference on Computer Systems* (Salzburg, Austria) *(EuroSys '11)*. Association for Computing Machinery, New York, NY, USA, 47–60. https://doi.org/10.1145/1966445.1966451

[35] unikraft.org [n.d.]. *Unikraft - Extreme Specialization for Security and Performance*.

[36] Dan Williams and Ricardo Koller. 2016. Unikernel Monitors: Extending Minimalism Outside of the Box. In *Proceedings of the 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'16)*. USENIX. https://www.usenix.org/conference/hotcloud16/workshop-program/presentation/williams

[37] Charles P Wright, Nikolai Joukov, Devaki Kulkarni, Yevgeniy Miretskiy, and Erez Zadok. 2005. Auto-pilot: A Platform for System Software Benchmarking.. In *USENIX Annual Technical Conference, FREENIX Track*. 175–188.