Automated Selection of Energy-efficient Operating System Configurations

Benedict Herzog
Friedrich-Alexander-Universität
Erlangen-Nürnberg (FAU)
benedict.herzog@cs.fau.de

Fabian Hügel
Friedrich-Alexander-Universität
Erlangen-Nürnberg (FAU)
fabian.huegel@fau.de

Stefan Reif
Friedrich-Alexander-Universität
Erlangen-Nürnberg (FAU)
reif@cs.fau.de

Timo Hönig
Ruhr-Universität
Bochum (RUB)
timo.hoenig@rub.de

Wolfgang Schröder-Preikschat
Friedrich-Alexander-Universität
Erlangen-Nürnberg (FAU)
wosch@cs.fau.de

ABSTRACT

Edge computing systems need to use their available resources efficiently. Operating systems and run-time systems offer numerous configuration parameters to fine-tune their behaviour, which are adjustable to balance the execution time and energy demand of applications. However, the number of parameters produces a vast space of possible configurations and the exact consequences on non-functional properties are often poorly documented. Thus, identifying efficient configurations proves challenging.

This paper presents Polar, an approach for the automated determination of energy-efficient configurations, as well as an implementation for Linux. Polar combines application profiles and system-level information to select efficient configurations dynamically and does not require application changes. Configurations are predicted by an oracle either based on linear models or neural networks. Our evaluation shows that Polar improves the mean energy efficiency by 11.5 % for typical applications.

ACM Reference Format:
Benedict Herzog, Fabian Hügel, Stefan Reif, Timo Hönig, and Wolfgang Schröder-Preikschat. 2021. Automated Selection of Energy-efficient Operating System Configurations. In *The Twelfth ACM International Conference on Future Energy Systems (e-Energy '21), June 28-July 2, 2021, Virtual Event, Italy.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3447555.3465327

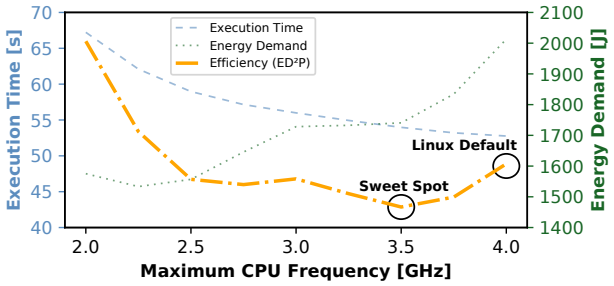

Figure 1: Execution time, energy demand, and energy efficiency (ED^2P) for compiling Linux on an Ubuntu system with different CPU frequency maxima.

1 INTRODUCTION

Edge computing sytems operate under resource constraints and therefore require all components to operate as efficient as possible [34, 36]. To this end, modern operating system offer a plethora of settings, such as valid operating frequency and buffer sizes, that need to be fine-tuned for achieving individual operational goals (i.e., performance improvements, low power draw). However, there is no universally optimal configuration—instead, the run-time effects of configurations often depend on application-specific aspects.

In this paper, we concentrate on an extensive exploration of system parameters of an operating system (i.e., Linux) and analyse their individual impact on the energy efficiency of different applications. Linux provides a set of "reasonably" predefined system configurations. Thus, Linux operates "well enough" on a large variety of different platforms, but rarely excels. Excellent run-time behaviour is only achieved in hand-tuned usage scenarios (e.g., high-performance computing). As hand-tuning operating system parameters is a labour-intensive (and by its complexity often infeasible) task [20] we further present Polar, an approach using an oracle that tunes operating system parameters to automatically improve the energy efficiency for arbitrary applications. Polar achieves an energy efficiency improvement of 11.5 % without changes to applications.

This paper is structured as follows. First, we present a motivating example in Section 2. Section 3 describes the Polar approach for energy efficiency savings. The implementations are presented in Section 4. Section 5 gives an evaluation of our implementations and Section 6 discusses possible further directions. Related work is discussed in Section 7 before we conclude the paper in Section 8.

e-Energy '21, June 28-July 2, 2021, Virtual Event, Italy

ACM ISBN 978-1-4503-8333-2/21/06…$15.00
https://doi.org/10.1145/3447555.3465327

2 MOTIVATING EXAMPLE

Operating systems (OSs) like Linux offer a broad set of configuration parameters[1] through several interfaces (e.g., the `sysfs` file system). In contrast to special-purpose operating systems, the default behaviour is optimised to work sufficiently well in the majority of environments and use cases. However, a well-working configuration seldom constitutes the optimal configuration for a given use case, which leaves the headroom for optimisations by tuning the currently active configuration [38].

Figure 1 illustrates such a use case, that is, compiling the Linux kernel on an unmodified Ubuntu system—illustrating heavy CPU, memory, and IO usage. This example shows the *CPU frequency maximum*, which is the maximum (boost) CPU frequency the Linux governor is allowed to set (4.0 GHz for our system). As expected, higher maximum frequencies yield shorter execution times. The energy demand, however, shows a more complex behaviour. Neither the lowest nor the highest maximum CPU frequency provides the lowest energy demand, but 2.5 GHz. The most energy-efficient configuration, considering both energy demand and execution time by means of the *energy-delay-squared product* metric (ED^2P), is at 3.5 GHz. The ED^2P is an efficiency metric, which includes the performance and energy demand of an application [8, 27]. The performance is squared to counterbalance the quadratic influence of the voltage on the energy demand.

In summary, traditional race-to-sleep (fastest execution, but high power draw) and crawl-to-sleep (lowest power draw, but slow) semantics [25, 26, 30] are not sufficient to choose the most energy-efficient configuration option, but a more extensive approach that considers workload characteristics is required. Furthermore, configuration parameters influence each other. For example, a CPU with deployed power cap may reduce its execution speed while executing the I/O scheduler, which influences the energy efficiency of the I/O component. A system, which reasons about the most energy-efficient configuration, thus needs to take into account all application's characteristics and all relevant components. The following section describes our Polar[2] approach for such a system.

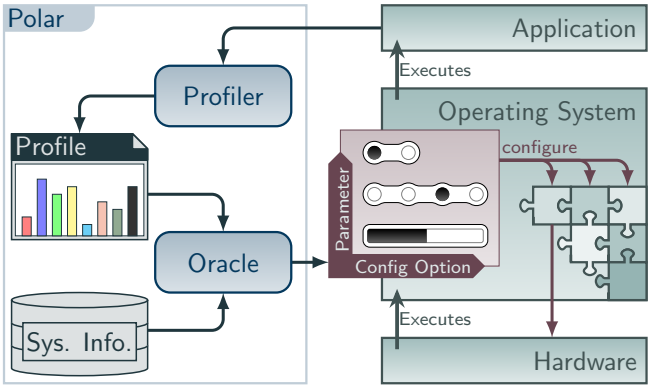

Figure 2: Polar adds energy awareness to operating systems while considering application and system characteristics. Polar's components are located in the light-blue frame.

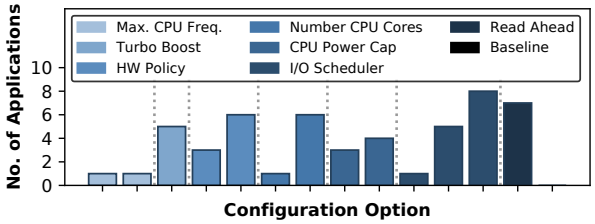

Figure 3: Histogram of configuration parameters optimality. The y-axis denotes how often selecting a specific configuration option resulted in the most energy-efficient application execution.

3 APPROACH

Our Polar approach, visualised in Figure 2, augments operating systems with energy awareness. Polar's components are located in the light-blue frame and comprise the following: First, a *system information* data set combines information about the influence of various application properties and system parameters on the energy efficiency. Second, a *profiler* gathers information about running applications and generates application *profiles*. Third, an *oracle* combines the information from the system data set and the application profile to select the most energy-efficient configuration set for a given application at run-time.

Our approach faces the inherent challenge of system complexity. All components have to account for interactions of a multitude of configuration options. The parameters span a multi-dimensional search space, which is too large for explicit enumeration. Only for tiny embedded platforms is a complete analysis considered possible, but even on smartphones, the complexity of hardware and software makes such approaches infeasible [10]. In consequence, appropriate measures for *generalisation* are required, which leads to an oracle that selects *good* configurations (in the sense that they are better than a baseline, including the overheads of the oracle itself), rather than the optimal one [20]. In the following, we present information about several oracle implementations that demonstrate the practicality of our approach. Furthermore, we present details about the gathered system information and the implementation of the profiler.

4 IMPLEMENTATION

This section describes the implementation of components introduced in the previous section. The implementation of Polar is built around the Linux operating system. However, the presented approach can also be implemented for other configurable runtimes and operating systems, and is not limited to Linux.

[1] This paper uses the following terminology: A "parameter" is an adjustable switch. Each parameter offers several "configuration options" (i.e., its state space). A (system) "configuration" selects a configuration option for each parameter.

[2] The name "Polar" refers to the observation that extreme configuration options are rarely efficient. Instead, the system has to find the optimal configuration in-between, considering both performance and power draw.

Name	Source	Description
realtime	syscall	Wall-clock time
usertime	syscall	Time spent in user mode
kerneltime	syscall	Time spent in kernel mode
inblocks	syscall	Block input operations
outblocks	syscall	Block output operations
instructions	perf	Retired instructions
cycles	perf	Total CPU cycles
llc_accesses	perf	Last-level cache accesses
llc_misses	perf	Last-level cache misses

Table 1: Profiling data constituting an application profile. The `perf` values are collected for both, user- and kernel-mode, separately.

4.1 System Information

The system information data consists of three different types of information, that is, the currently applied configuration, the executed application's profile, and the corresponding execution time and energy demand. We gathered this information by conducting time and energy measurements for a rich set of applications (cf. Section 4.2) and different configurations. This data allows the oracle to reason about the effect of a configuration change on the energy efficiency for each application individually.

Linux provides several interfaces for configuration changes during run-time. We utilise the `sysfs` interface, which is the most general and important configuration interface. The `sysfs` interface provides a broad range of low-level configuration parameters (e.g., enable/disable Turbo Boost or apply a RAPL power cap). Figure 3 shows all considered configuration parameters, which cover the CPU, memory, and block I/O subsystem. Hence, our approach is not limited to a single subsystem but can utilise the headroom of different subsystems. In total, we consider seven parameters with 22 different configuration options. As different applications make use of different subsystems, the respective energy efficiency savings lie in one subsystem or the other. In order to identify relevant subsystems, the application profiles must represent the behaviour of an application.

4.2 Application Profiles

For application profiles, we utilise software performance counters obtained from the Linux kernel (i.e., `getrusage()` for I/O intensity and `clock_gettime()` for time information) and hardware performance counters (`perf_event_open()`) which, in summary, describe an application's behaviour. Table 1 gives an overview of the utilised performance counters and their source. All `perf`-related values are collected separately for user- and kernel-mode. Thereby, the time-related information allows a differentiation of applications with high system activity compared to applications without much interaction with the operating system. The block input and output operations enable reasoning about the usage intensity of the block I/O subsystem. In addition, instructions, cycles, and last-level cache statistics provide information about code- and data-locality on the one side and CPU- and memory-intensity on the other side. This leads to a total of 13 counters constituting an application profile.

4.3 Oracle

The oracle's task is to combine the system information and application profiles to predict an energy-efficient configuration on a per-application basis. We implemented two oracle types: (a) based on linear models and (b) based on a neural network.

Linear Model. The input for the model is an application profile. The output of the oracle predicts the most-efficient configuration option. The main limitation of linear models in our case is their inability to work with categorical data. Hence, we trained linear models for each parameter with numeric value space (e.g., the maximum CPU frequency) and round to the next available configuration option. In total, we implemented and trained linear models for three configuration parameters, that is, the number of CPU cores, the maximum CPU frequency, and the CPU power cap. For all linear models, we utilise the *scikit-learn* python framework [35].

Neural Network. Neural networks are capable of modelling complex correlations for big search spaces [14, 21, 37]. This capability makes it a suitable implementation candidate for an oracle. For the implementation of our neural network, we rely on the TensorFlow framework [39] and utilise feed-forward neural networks.

The application profile is directly used as input for the neural network as it solely consists of numerical values. The output layer of the neural network consists of one output value per configuration option. As our configuration set consists of 22 configuration options, the output layer consists of 22 outputs. The output layer comprises a *softmax* layer, that is, each output value lies between 0 and 1 and represents the probability that the corresponding configuration option is the most energy efficient option. The output value with the highest probability constitutes the predicted configuration option.

In order to determine the network's architecture, we conducted a *random search* on the hyperparameters as proposed by Bergstra et al. [5]. Thereby, we varied the number of layers (3-4), neurons per layer (32-256), activation function (`relu`, `sigmoid`), optimiser (`adam`, `sgd`, `rmsprop`), and dropout rate (0.1-0.5). Furthermore, we used different numbers of epochs (100-1000). In total, 100 network candidates were trained using the training set and the validation set was used to decide the final network architecture. The finally selected neural network consists of three feed-forward layers with 160 (`relu`), 256 (`sigmoid`), and 224 (`relu`) neurons, respectively. During training, a dropout value of 0.36 was used. In total, 419 epochs of training using the `adam` optimiser were conducted.

5 EVALUATION

The evaluation of Polar consists of three parts: (a) we analyse the potential energy efficiency improvements, (b) we evaluate the prediction quality of our oracles, and (c) we analyse the oracle overheads.

5.1 Evaluation Setup

Our evaluation setup consists of three identical desktop systems hosting Intel Core i5-8400 CPUs with a base frequency of 2.8 GHz and a maximum frequency of 4.0 GHz. The operating system is Ubuntu 18.04 LTS (Linux v4.15). We conducted our energy demand measurements with Microchip MCP39F511N power monitor boards.

Training Set		Validation Set		Evaluation Set	
NAS Parallel Benchmarks	21	NAS Parallel Benchmarks	3	Git Compilation	2
Flexible I/O Tester	7	Compress, Encrypt, SHA256	2	Mandelbrot Image, FFmpeg	3
Linux Kernel Compilation	2	Linux Kernel Compilation	1	SQLite Insert	1
Small File Handling	1	Prime Number Generator	2	xz Compression	2
		Stockfish Chess Engine	2	Gaussian Mixture Model	2
Total Applications:	**31**	**Total Applications:**	**10**	**Total Applications:**	**10**

Table 2: Overview of the application sets used for training, validation, and evaluation. The application selection is designed to cover both: subsystem-specific and general applications.

The MCP39F511N measures whole-system power as drawn from the wall socket, with a sampling rate of 400 Hz and a measuring error below 0.1 % [29]. We use the energy-delay-squared product (ED^2P) metric, which combines execution time and energy demand, to analyse the energy efficiency [8]. The ED^2P is especially suited to analyse systems with applied DVFS [8, 27].

We execute one (discarded) warm-up round and five iterations for each application and use the respective mean of these iterations for the energy demands and application profiles. Between iterations, we flush caches and I/O buffers to ensure identical initial conditions. Table 2 summarises the applications used for training, validation, and evaluation.

5.2 Potential Energy Efficiency Improvements

Prerequisite to evaluate our approach is to analyse the potential energy efficiency improvements. For our application sets an ideal oracle, always selecting the optimal configuration option, can improve the energy efficiency by 13.3 % for the evaluation set (13.5 % validation set, 22.9 % training set). The most efficient option varies per application. The histogram in Figure 3 illustrates that each parameter is useful for some applications, and no single configuration is consistently optimal across workloads. Each bar denotes one configuration option and bars sharing the same color belong to the same parameter. Configuration options, which were never optimal, are omitted in the histogram. Thus, 13 out of 22 configuration options are at least once optimal.

However, changing the configuration not only has the potential to improve energy efficiency. In the evaluation set, 47 % of the tested configuration options degrade the energy efficiency. Furthermore, the geometric mean of these potential degradations (39 %) is much higher than potential improvements (9 %). This is in line with the expectation that a general-purpose operating system like Ubuntu is reasonably well optimised for generic workloads. Consequently, oracles must choose their predictions very carefully.

5.3 Oracle Predictions

Figure 4 compares the achieved energy efficiency for all three linear models and the neural network for the evaluation set. Notably, all models have a positive impact on the energy efficiency despite the potential for degradations analysed in the previous section. Hence, all implementations improve the energy efficiency of our system under test. However, the neural network achieves the highest improvements, which is why it is analysed in more depth in the following section.

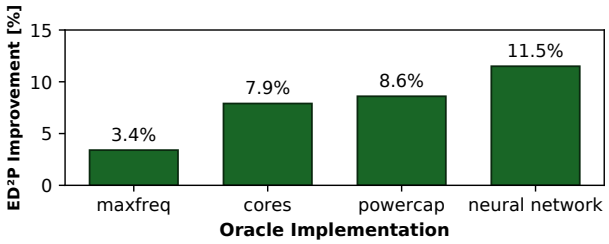

Figure 4: ED^2P savings for the evaluation set per oracle implementation. Although all implementations have a positive impact, the neural network yields the greatest savings.

5.4 Neural Network Predictions

Figure 5 shows the energy efficiency results for all applications executed with the configuration as predicted by the neural network, partitioned into training, validation, and evaluation set. The blue (training), yellow (validation), and green (evaluation) bars show improvements and the red bars a degradation in energy efficiency. For each application, the maximum potential energy efficiency improvement is shown as a light-grey bar (i.e., the potential improvement when selecting the optimal configuration option).

For the training set, the oracle rarely predicts a configuration with degraded energy efficiency. In fact, only for three out of 31 applications, the energy efficiency is marginally reduced compared to the baseline. For the rest of the applications, the oracle predicts a configuration with improved energy efficiency. For both, the validation and evaluation set, the energy efficiency is improved in all cases. The geometric mean of the energy efficiency for the training set is improved by 8.9 %, for the validation set by 11.7 %, and for the evaluation set by 11.5 % compared to the baseline. Especially for the validation and evaluation set, this exploits a significant amount of the maximum possible improvement of 13.5 % and 13.3 %, respectively. These results show that considerable improvements in the energy efficiency are buried in the configuration of our operating systems, which can be exploited automatically without changes to applications.

5.5 Oracle Evaluation

Both oracle types not only differ in prediction performance, but also in execution time and energy demand for training and prediction itself. This section analyses these costs for both types. Furthermore, the section gives an estimation of the break-even point,

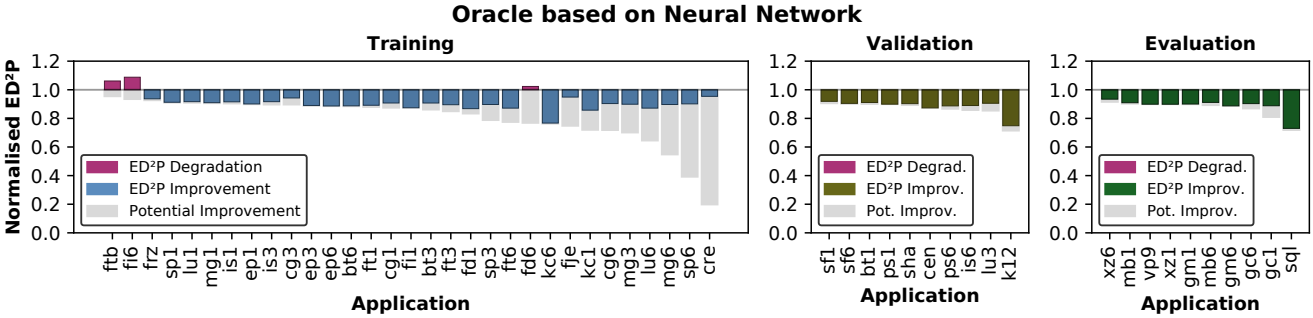

Figure 5: Energy efficiency in terms of ED^{2}P normalised to the baseline for all applications for the configuration chosen by our oracle based on the neural network. The left side shows the training set, the graph in the middle the validation set, and the right side shows the evaluation set. Additionally, the potential energy efficiency improvements are illustrated.

when predictions save more energy than spent on training and prediction overheads. All measurements are executed on the same setup as described in Section 5.1.

Training Overhead. Training the neural network needs 3.4 s and 110 J (52 J for the CPU measured with RAPL). The hyperparameter optimisation (i.e., testing 100 neural networks) needs a total of 96.6 s and 6229.8 J. However, the hyperparameter optimisation only needs to be conducted once and the result can be reused for later trainings.

The training time for one linear model is 0.26 s and the energy demand 8.4 J (RAPL: 4.24 J). As expected, this is much lower than training the neural network. The training of the other linear models shows a similar execution time and energy demand behaviour.

Prediction Overhead. Although the overhead for hyperparameter optimisation and training a neural network is higher compared to a linear model, neural networks can be efficiently executed. The execution time for the neural network is too short for measurements with the measurement device described in Section 5.1, which is the reason why we use the RAPL interface to determine the energy demand for executing the neural network once. One prediction—executed on the CPU—needs 36 ms and 0.56 J.

The execution time for the linear model is below 1 ms and we can neither use the energy device nor RAPL to reliably determine the energy demand, because the execution time is too short.

Break-even Analysis. The results for the break-even analysis depend significantly on the application and its potential energy efficiency improving. For a fair analysis, we use the single-threaded kernel compilation (`kc1`) as a common real-world application. As depicted in Figure 5, the `kc1` application shows average behaviour in terms of potential and achieved energy-efficiency improvements. Thus, it reflects neither a best-case nor a worst-case scenario but is an example of a typical application.

We assume that costs for training and hyperparameter optimisation occur once and prediction costs occur once per execution of the application. This reflects that the oracle is trained once and predicts a configuration every time executing an application. We use the energy-demand measurements from the measurement device where possible and RAPL measurements plus a static system power otherwise. For predictions of the linear model we linearly scale the energy demand of the neural network to 1 ms, which is an upper bound of the execution time for the linear model.

With these assumptions, the `kc1` application must be executed 19 times until the savings of the improved configuration exceed the hyperparameter optimisation and training costs using the neural network. If the hyperparameter optimisation is excluded and only the training of the neural network is considered, the oracle saves more energy with the first execution of the application. The linear model likewise saves more energy than spent with the first execution of the application. However, for the linear model there exist applications where no or less energy efficiency savings are possible due to the restricted configuration space.

These results show that the configuration savings compensate the training costs in short amounts of time—in particular in edge scenarios where tasks are executed repeatedly. The benefit of improved energy efficiency quickly outweighs the costs of Polar related to the training and application of the model. Furthermore, we expect that training and execution costs for neural networks will reduce in the future due to hardware acceleration in modern CPUs and dedicated hardware like Tensor Processing Units [23]. Hence, our approach not only achieves energy efficiency savings without application changes, but does so in a sustainable way.

6 FUTURE WORK

Although our approach and implementation show good results, Polar offers room for future improvement.

Configuration Interdependencies. Our current implementations consider all parameters, but eventually select only one parameter to optimise. It is possible that a combined change of several parameters yield better improvements. However, an early analysis showed no significant potential efficiency improvements, but simultaneously a significantly more complex prediction problem.

Hardware Dependency. Our approach and implementation is not limited to a specific hardware platform. The trained models, however, are specific to the hardware platforms contained in the training set. To support a new hardware platform corresponding training data must be included and the models must be (re-)trained.

On-line Learning. Our approach uses pre-trained models that have to be created before running applications. Future work can learn or adapt the model on-line which renders the dedicated training phase unnecessary. Training the model on-line further enables the system to automatically adapt to software updates and hardware upgrades.

Application Concurrency. Our current implementations act on the assumption that only one application is running simultaneously (i.e., the models use one application profile to predict configuration changes). This is sufficient for scenarios with only one major application running simultaneously, and other workload (e.g., background daemons) are negligible. For future work we plan to extend our approach for multiple concurrently running applications.

7 RELATED WORK

Tuning of system parameters for various operational goals has been an active field of research, particularly, since adaptive hardware components [12] are available [9]. The efficient use of hardware offerings strictly depends on software measures at operating system level to utilise available hardware features in the best manner for given applications. The dynamic adaptation during run-time of hardware components eventually require software means to successfully reach certain operation goals [1, 20, 32].

The trade-off between performance and energy-efficiency has been intensively studied and explored [4, 7, 17, 18, 25, 26, 40] as the control over operating voltage and power management has been broadly transferred from the hardware and firmware level towards the operating system. With this increased responsibility at OS level, Benini et al. [3] proposed a feedback-based control scheme for system-level dynamic power management. The work monitors system activities to adapt the configuration of sleep delays accordingly. Our work, in contrast, considers bank shots (i.e., also configuration parameters unrelated to power management) to control the system in a desirable manner (i.e., improve energy efficiency). Our approach further benefits by advances of the past decades, as additional parameters with more fine-grained control (e.g., DVFS [25, 26], RAPL [11]) are available and provide active control of measures (e.g., buffers of block device) that indirectly impact energy efficiency [2].

Operating system designers acknowledged the importance of energy as a resource just as important as time [31, 42]. With the consideration of energy to be *just another resource* that is available to the operating system, operating system internals (i.e., locking mechanisms [15]) were adapted to factor in this new paradigm to implement energy-aware operating systems [6, 33]. To take advantage of available parameters, several approaches toward self-tuning of system parameters have been proposed [16, 22]. In contrast to previous approaches, our work is first to explore large solution spaces.

Applying machine learning to improve energy efficiency of computing systems has been subject to research in the context of adapting data-centres [6], cloud computing [13], and big data applications [19]. Our work successfully applies machine learning techniques at a different, much lower level of abstraction. Hence, our approach is orthogonal to this related research. As energy efficiency is subject to the individual applications, our proposed solution is designed to support legacy software and does not require existing program code to be altered. This is necessary with related approaches [24, 41].

Machine learning techniques have also been applied at hardware level to guide microarchitectural optimisations [28]. Hardware resources must fulfill a broad range of technical requirements (e.g., DRAM timing requirements) and determining a good configuration involves much engineering power at design time. Furthermore, the hardware can adapt to the currently executed software at runtime to further optimise its efficiency. For both approaches machine learning can guide optimisations. However, the optimised parameters are not (all) exposed to the operating system and hence constitute an orthogonal set of optimisable parameters.

8 CONCLUSION

This paper has presented the Polar approach for automated energy-aware operating system reconfiguration for edge computing. It selects energy-efficient configurations based on application profiles and system-level information. Internally, it utilises either a neural network or linear models as oracles to manage the complexity of vast configuration spaces. The oracle implementations allow energy efficiency improvements of up to 11.5 %. Polar accomplishes these improvements without changes to applications and disadvantages for users. Thus, Polar helps to exploit the available energy efficiency improvements buried in the configuration of operating systems in general and Linux in particular.

AVAILABILITY

The source code for Polar is available at https://gitlab.cs.fau.de/i4/pub/polar under an open-source license.

ACKNOWLEDGEMENT

This work was partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 146371743 – TRR 89 "Invasive Computing", and under grant no. SCHR 603/15-2 ("eLARN") and 603/10-2 ("COKE2") and from the Bundesministerium für Bildung und Forschung (BMBF, Federal Ministry of Education and Research) in Germany for the project AI-NET-ANTILLAS 16KIS1315.

REFERENCES

[1] Luiz André Barroso and Urs Hölzle. 2007. The case for energy-proportional computing. *Computer* 40, 12 (Dec. 2007).

[2] Frank Bellosa. 2000. The Benefits of Event-Driven Energy Accounting in Power-Sensitive Systems. In *Proceedings of the 2000 ACM SIGOPS European Workshop „Beyond the PC: New Challenges for the Operating System" (EW '00)*. ACM, 37–42.

[3] Luca Benini, Alessandro Bogliolo, Stefano Cavallucci, and Bruno Riccó. 1998. Monitoring system activity for OS-directed dynamic power management. In *Proceedings of the 1998 International Symposium on Low Power Electronics and Design (ISLPED '98)*. ACM, 185–190.

[4] Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. 2000. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 8, 3 (June 2000), 299–316.

[5] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, 1 (2012), 281–305.

[6] Josep Ll. Berral, Íñigo Goiri, Ramón Nou, Ferran Julià, Jordi Guitart, Ricard Gavaldà, and Jordi Torres. 2010. Towards Energy-Aware Scheduling in Data Centers Using Machine Learning. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking (e-Energy '10)*. ACM, 215–224.

[7] David Brooks and Margaret Martonosi. 2001. Dynamic thermal management for high-performance microprocessors. In *Proceedings of the 2001 Symposium on High-Performance Computer Architecture (HPCA '01)*. IEEE, 171–182.

[8] David M. Brooks, Pradip Bose, Stanley E. Schuster, Hans Jacobson, Prabhakar N. Kudva, Alper Buyuktosunoglu, John-David Wellman, Victor Zyuban, Manish Gupta, and Peter W. Cook. 2000. Power-Aware Microarchitecture: Design and Modeling Challenges for next-Generation Microprocessors. *IEEE Micro* 20, 6 (2000), 26–44.

[9] Thomas D Burd, Trevor A Pering, Anthony J Stratakos, and Robert W Brodersen. 2000. A dynamic voltage scaled microprocessor system. *IEEE Journal of Solid-State Circuits* 35, 11 (November 2000), 1571–1580.

[10] Markus Buschhoff, Daniel Friesel, and Olaf Spinczyk. 2018. Energy Models in the Loop. *Procedia Computer Science* 130, 1063–1068.

[11] Howard David, Eugene Gorbatov, Ulf R Hanebutte, Rahul Khanna, and Christian Le. 2010. RAPL: memory power estimation and capping. In *Proceedings of the 2010 International Symposium on Low-Power Electronics and Design (ISLPED '10)*. IEEE, 189–194.

[12] Michael J DeLuca and Mario A Rivas. 1992. Computing system with selective operating voltage and bus speed. U.S. Patent 5,086,501.

[13] M. Demirci. 2015. A Survey of Machine Learning Applications for Energy-Efficient Resource Management in Cloud Computing Environments. In *Proceedings of the 2015 International Conference on Machine Learning and Applications (ICMLA '15)*. 1185–1190.

[14] Richard Evans and Jim Gao. Jul 20, 2016. DeepMind AI reduces Google data centre cooling bill by 40%. *Deepmind Blog* (Jul 20, 2016). deepmind.com/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-40/, Acc. 2020-01-15.

[15] Babak Falsafi, Rachid Guerraoui, Javier Picorel, and Vasileios Trigonakis. 2016. Unlocking energy. In *Proceedings of the 2016 Annual Technical Conference (ATC '16)*. USENIX, 393–406.

[16] Krisztián Flautner and Trevor Mudge. 2002. Vertigo: automatic performance-setting for Linux. *ACM SIGOPS Operating Systems Review* 36, SI (2002), 105–116.

[17] Matthew Garrett. 2007. Powering down. *Queue* 5, 7 (Nov. 2007), 16–21.

[18] Dirk Grunwald, Charles B Morrey III, Philip Levis, Michael Neufeld, and Keith I Farkas. 2000. Policies for dynamic clock scheduling. In *Proceedings of the 2000 Symposium on Operating Systems Design and Implementation (OSDI '00)*. USENIX, 1–14.

[19] Álvaro Brandón Hernández, María S Perez, Smrati Gupta, and Victor Muntés-Mulero. 2018. Using machine learning to optimize parallelism in big data applications. *Future Generation Computer Systems* 86 (2018), 1076–1092.

[20] Benedict Herzog, Stefan Reif, Fabian Hügel, Timo Hönig, and Wolfgang Schröder-Preikschat. 2021. Poster: Towards Automated System-Level Energy-Efficiency Optimisation using Machine Learning. In *12th ACM International Conference on Future Energy Systems (e-Energy'21)*. 1–2.

[21] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29, 6 (2012), 82–97.

[22] Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin Rinard. 2011. Dynamic knobs for responsive power-aware computing. *ACM SIGPLAN Notices* 46, 3 (2011), 199–212.

[23] Norman Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-datacenter performance analysis of a tensor processing unit. In *44th Annual International Symposium on Computer Architecture (ISCA'17)*. 1–12.

[24] Mahmut Kandemir, Narayanan Vijaykrishnan, Mary Jane Irwin, and Wu Ye. 2000. Influence of compiler optimizations on system power. In *Proceedings of the 2000 Design Automation Conference (DAC '00)*. ACM, 304–307.

[25] Etienne Le Sueur and Gernot Heiser. 2010. Dynamic Voltage and Frequency Scaling: The Laws of Diminishing Returns. In *Proceedings of the 2010 International Conference on Power Aware Computing and Systems (HotPower'10)*. USENIX Association, 1–8.

[26] Etienne Le Sueur and Gernot Heiser. 2011. Slow Down or Sleep, That is the Question. In *Proceedings of the USENIX Annual Technical Conference (ATC'11)*. USENIX Association, 1–6.

[27] Alain Martin, Mika Nyström, and Paul Pénzes. 2002. ET^2: A Metric for Time and Energy Efficiency of Computation. In *Power Aware Computing*, Robert Graybill and Rami Melhem (Eds.). Springer US, 293–315.

[28] Jose F. Martinez and Engin Ipek. 2009. Dynamic Multicore Resource Management: A Machine Learning Approach. *IEEE Micro* 29, 5 (2009), 8–17.

[29] Microchip. [n.d.]. Microchip MCP39F511 data sheet. URL: https://microchip.com/wwwproducts/en/MCP39F511. Acc. 2020-01-12.

[30] Akihiko Miyoshi, Charles Lefurgy, Eric Van Hensbergen, Ram Rajamony, and Raj Rajkumar. 2002. Critical Power Slope: Understanding the Runtime Effects of Frequency Scaling. In *16th International Conference on Supercomputing (ICS 2002)*. 35–44.

[31] Rolf Neugebauer and Derek McAuley. 2001. Energy is just another resource: energy accounting and energy pricing in the Nemesis OS. In *Proceedings of the 2001 Workshop on Hot Topics in Operating Systems (HotOS '01)*. IEEE, 67–72.

[32] Padmanabhan Pillai and Kang G Shin. 2001. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the 2001 Symposium on Operating Systems Principles (SOSP '01)*. ACM, 89–102.

[33] Arjun Roy, Stephen M Rumble, Ryan Stutsman, Philip Levis, David Mazières, and Nickolai Zeldovich. 2011. Energy management in mobile devices with the Cinder operating system. In *Proceedings of the 2011 European Conference on Computer Systems (EuroSys '11)*. ACM, 139–152.

[34] Mahadev Satyanarayanan. 2017. The Emergence of Edge Computing. *IEEE Computer* 50, 1 (Jan. 2017), 30–39.

[35] Scikit-learn. [n.d.]. Scikit-learn machine learning framework. https://scikit-learn.org. Acc. 2020-01-12.

[36] Weisong Shi and Schahram Dustdar. 2016. The Promise of Edge Computing. *IEEE Computer* 49, 5 (May 2016), 78–81.

[37] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature* 529, 7587 (2016), 484–489. https://doi.org/10.1038/nature16961

[38] David Snowdon, Etienne Le Sueur, Stefan Petters, and Gernot Heiser. 2009. Koala: A Platform for OS-level Power Management. In *Proceedings of the 4th European Conference on Computer Systems (EuroSys'09)*. ACM, 289–302.

[39] TensorFlow. [n.d.]. TensorFlow machine learning framework. https://tensorflow.org. Acc. 2020-01-12.

[40] Andreas Weissel and Frank Bellosa. 2002. Process cruise control: event-driven clock scaling for dynamic power management. In *Proceedings of the 2002 Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES '02)*. ACM, 238–246.

[41] Tomofumi Yuki and Sanjay Rajopadhye. 2013. Folklore confirmed: compiling for speed = compiling for energy. In *Proceedings of the 2013 Workshop on Languages and Compilers for Parallel Computing*. Springer, 169–184.

[42] Heng Zeng, Carla S Ellis, Alvin R Lebeck, and Amin Vahdat. 2002. ECOSystem: managing energy as a first class operating system resource. In *Proceedings of the 2002 Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '02)*. ACM, 123–132.