Dynamic Voltage-Frequency Optimization using Simultaneous Perturbation Stochastic Approximation

Evgenii Bogdanov
St. Petersburg State University
St. Petersburg, Russia
evgenij.bogdanov.1999@gmail.com

Alexander Bozhnyuk
St. Petersburg State University
St. Petersburg, Russia
bozhnyuks@mail.ru

David Bykov
St. Petersburg State University
St. Petersburg, Russia
bykov.david@gmail.com

Stanislav Sartasov
Software Engineering Department
St. Petersburg State University
St. Petersburg, Russia
Stanislav.Sartasov@spbu.ru

Anna Sergeenko
Software Engineering Department
St. Petersburg State University, IMPE RAS
St. Petersburg, Russia
a.sergeenko@spbu.ru

Oleg Granichin
Faculty of Mathematics and Mechanics,
Research Laboratory for Analysis
and Modeling of Social Processes
St. Petersburg State University, IMPE RAS
St. Petersburg, Russia
o.granichin@spbu.ru

***Abstract*—It was previously shown that simultaneous perturbation stochastic approximation can be used for developing general purpose dynamic voltage-frequency scaling (DVFS) governor for Android OS. We lay down a theoretical foundation for this approach and show in several experiments that our algorithm performance is on par with the commonly used Android DVFS governors. Links to complete results and source code are also provided.**

***Index Terms*—Android, energy consumption, simultaneous perturbation stochastic approximation, SPSA, dynamic voltage frequency scaling, DVFS**

I. Introduction

Smartphones and tablets became an essential part of modern life. While they provide high quality digital services, the most limiting factor of their usage is their energy consumption. With more programs are running and peripherals turned on a device battery is drained faster. The importance of this problem is evident to anybody left with a discharged phone when an important call or piece of information was needed.

Battery charge is limited by definition, therefore one needs to prolong it as much as possible. Various approaches could be taken together to address this issue. Advances in electrochemistry result in a more capacious batteries. Hardware is designed to support energy awareness and to decrease its energy consumption when not actively in use. Applied software implements “green” development practices like energy profiles or energy refactorings [1]. An end user might consider turning off unused features or uninstall draining applications. Summary effect of those means could be considerable.

This work was supported by the Russian Science Foundation (Project No. 21-19-00516).

In this paper, we would like to address an operating system (OS) level of energy awareness. Modern embedded processors support dynamic voltage frequency scaling (DVFS) which is a way to decrease or increase its operating voltage and/or frequency to optimize energy consumption. Usually operating systems have a direct control over CPU frequencies in a form of DVFS governors — OS modules which analyze the current CPU load and scale its frequencies accordingly on a per-chip or per-core basis. A significant amount of DVFS governors was developed over the years [2].

In a sense, DVFS governors save battery power by making applications run as slow as perceivable comfortable. By introducing a target metric DVFS workload could be interpreted as an optimization task. To solve it, simultaneous perturbation stochastic approximation (SPSA) may be considered. SPSA may be viewed as a random search technique, and on average the algorithm will nearly follow the steepest descent direction [3]. While it was previously used to track changing conditions in various computer science tasks and react accordingly [4], [5], its application for DVFS purposes in modern smartphones was not previously investigated. Thus, main contribution of this paper is a new algorithm for dynamic CPU frequency changing based on the SPSA approach and its comparison with other widespread DVFS-algorithms used in Android OS by default, like OnDemand and Interactive.

This paper is organized as follows. In Section II, an overview of DVFS algorithms used in modern smartphones is given along with a description of SPSA approach and a review of previous work on related topics. Section III contains a description of the proposed DVFS governor architecture. Experimental methodology is outlined in Section IV, experiments themselves are analyzed in Section V and Section VI concludes the paper.

II. Background

A. Dynamic Voltage Frequency Scaling in Android OS

A number of DVFS algorithms already exists for Android OS and is generally available in a smartphones by default. Some of them were inherited from Linux OS, while others were developed specifically for Android OS [2]:

a) OnDemand: This governor increases the CPU operating frequency as soon as the processor is loaded with computation tasks to keep system responsive. When a certain CPU load threshold ($\sim 80\%$) is reached, the governor increases the frequency to the maximum until the processor load decreases.

b) PowerSave: In order to reduce CPU energy consumption this governor reduces CPU frequency to the minimal available frequency and keeps it at a constant level.

c) Performance: This governor works exactly the opposite it constantly uses the maximum available processor frequency for the best performance.

d) Conservative: When there is no load this governor uses the lowest available frequency, and when there's an activity on a CPU it gradually increases the frequency decreasing it back when CPU is idle.

e) Interactive: The governor is designed for delay-sensitive workloads such as interactive user interfaces. Operating frequency is dependant on the load, and the load check is event-driven instead of timer-driven and occurs when the user begins to interact with the system.

While there is a large number of DVFS algorithms created by standalone developers to enhance the characteristics of the default algorithms, additional approaches are covered in literature.

To handle DVFS one may create a regression model based on the data received from the processor (temperature and power) and train it to set the required DVFS parameters for each situation. Authors report energy savings of 10% on average compared to default DVFS policies.

The algorithm using counter propagation neural networks analyzes the current system behavior (performance, instructions, cache status) and selects the necessary frequency [6]. For a multi-core processor, the mechanism for selecting the optimal frequency is launched for each core separately. When measuring power consumption for a single-core mode, a saving of 5% to 20% of energy was obtained with a performance loss of 10% and from 9% to 42% of energy saving with a performance loss of 30%. In multi-core experiments, the approach allowed saving from 2.3% to 8.8% of energy with a limit a performance loss of 10% and from 3.9% to 22% of energy with a limit a performance loss of 30%. “Long Short-term Memory” of the recursive network may be used to create DVFS governor [7]. It is important to make a correlation only for short-term data due to the gradient attenuation problem. This network architecture reduces the power consumption of processors by a maximum of 19% compared with the existing DVFS. Using reinforcement learning to predict CPU load allowed to achieve energy savings of 22% on average [8].

Another class of algorithms is based on priming the governor for a specific user activity before actual smartphone usage. For example, one could select the optimal frequency value based on user transactions(for example, touching the screen or scrolling), ending with the last update of the display [9]. While using the device, if there is already a known transaction, the previously calculated processor frequency is set. Energy savings when implementing this approach ranged from 10% to 36%, while not affecting the quality of user experience.

Another class of governors calculates optimal frequency by some formula or by a specific process based on the current state of the system. For example, the main criterion for changing the frequency could be the processor temperature. If a certain temperature threshold is exceeded, a frequency-history window is filled and CPU momentary frequency is set [10]. At the same time, there are two optimization options: performance and power consumption. The resulting algorithm reduces power consumption by an average of 12.7% when using the power optimization option, and by 6.7% when using the performance optimization option. In the latter case, execution time increases on average by 6.3%. As another example, power consumption might be optimized by using DVFS along with sensitive processor bandwidth control [11]. This approach determines optimal number of active CPU cores as well as their frequency, and controls allocated bandwidth to achieve best working state of the CPU in terms of performance and energy saving. Such a scheme allows to get battery savings of 14% and 23% for more and less dynamic workloads, respectively.

B. Simultaneous Perturbation Stochastic Approximation

A number of control problems can be solved by finding the extrema of empirical functional which models behavior of a system in question. In our case, CPU workload might be evaluated for some time to determine what is the best frequency to use given a set of available frequencies and quality criteria.

More formally, let $F_t(x, w)$ be a function of discrete time t, some parameter x and randomised vector w. Define medium risk functional as

$$f_t(x) = E_w F_t(x, w)$$

and the minimum point of $f_t(x)$ as

$$\theta_t = \arg\min_x f_t(x).$$

Thus our purpose is to build the sequence of estimations $\{\hat{\theta}_n\}$ such that $||\hat{\theta}_n - \theta_t|| \rightarrow \min$ based on observations of the random variables $F_t(x_n, w_n)$, n = 1, 2,..

We define a *momentary trial perturbation* as a sequence of observed uniformly symmetrically distributed independent random vectors Δ_n with covariance matrices

$$cov\{\Delta_n \Delta_j^T\} = \delta_{nj}\sigma_\Delta^2 I,$$

where $\delta_{nj} \in \{0, 1\}$ is the Kronecker symbol, $0 < \sigma_\Delta < \infty$. Bernoulli random vectors are frequently used as such simultaneous trial perturbation as vector coordinates Δ_n are independent from one another and have equiprobable values of ± 1.

It was shown that under noisy observations it is possible to use the following algorithm

$$\hat{\theta}_n = \hat{\theta}_{n-1} - \frac{\alpha_n}{\beta_n}\Delta_n y_n,$$

to build a minimum point estimation sequence for a functional $F(x)$ without significant loss of convergence rate [12]. At each iteration this algorithm uses only a single noisy evaluation of empirical function:

$$y_n = F(\hat{\theta}_{n-1} + \beta_n \Delta_n, w_n^+) + v_n.$$

$\{\alpha_n\}$ and $\{\beta_n\}$ here are sequences of non-negative numbers conforming to some conditions, w_n^+ is a stochastic perturbation vector for y_n^+ observation, v_n^+ is an arbitrary external noise during the observation. This recurrent procedure is called *simultaneous perturbation stochastic approximation* (SPSA) because it inseparably contains a randomized trial perturbation which is simultaneous in all coordinates. It is also used to set a direction of the next estimation change and to select a new evaluation point. Among the conditions for consistency of estimates we specifically set out a condition for weak correlation between trial perturbation $\{\Delta_n\}$ and sequences of indeterminacies $\{w_n\}$ and $\{v_n\}$ as the most important. While mean squared convergence rate of this algorithm is not the best one compared to other algorithms of its class, from a practical perspective it is of use because in real time systems optimizations and adaptive control problems two or more observations with noises independent from Δ_n are not available.

A general flow of SPSA algorithm for system control can be summarized as follows:

1) Define an empirical functional $F(x)$.
2) Make an initial optimal estimate of $\hat{\theta}_0$.
3) Perturb a current optimal estimate.
4) Obtain a new noisy observation of F and update a current optimal estimate $\hat{\theta}_n$.
5) Go to Step 3.

III. Algorithm

A. Description

Our proposed model is based on a number of assumptions. Let's define a CPU workload as a percentage of time CPU or its core spent on executing processes. Our first assumption is that CPU resources are frequently underutilized, so the workload rarely achieves 100%. Broadly, CPU frequency defines an amount of work it can do in a selected time interval, so by decreasing frequency we increase the amount of time the same workload items take to execute.

The general idea of our algorithm is to predict the frequency of the next time interval by the current state of the CPU in terms of a workload, so that workload level is as close to a preliminarily selected value as possible. This value should not be close to a 100% so it would still possible to allocate additional computational time if we underestimated the load, and not too close to 0% as it increases operating frequency and energy consumption.

Secondly, most of the modern smartphone processors, specifically those supporting big.LITTLE architecture, operate only on a fixed set of frequencies $Freq$, so our continuous frequency estimate should be rounded to the nearest available value.

We define our model as

$$F(f) = 2^{((\text{workload}(f) - \lambda)/2)} + \gamma 1.5^{\text{table}(f)},$$

where $\text{workload}(f)$ is CPU workload as obtained from system metrics, λ is the target frequency, $\text{table}(f)$ is the number of a frequency in a available CPU frequencies table sorted from lowest to highest. The first term is a penalty for too little workload compared to the reference value λ. It is important to mention that since the same computational load will take less time with increasing frequency, the function $\text{workload}(f)$ is non-increasing. The second term being monotonically rising is introduced as a penalty for using too high frequency from the set.

To update the operating frequency we calculate

$$f_n = \mathscr{P}(\hat{f}_{n-1} + \beta\Delta_n),$$

where $\mathscr{P}$ is the projection to $Freq$, $\hat{f}_{n-1}$ is the current frequency estimate, β is the SPSA step size parameter. Then, to make a new frequency estimate we calculate

$$\hat{f}_n = \mathscr{L}(\hat{f}_{n-1} - \frac{\alpha}{\beta}\Delta_n y_n),$$

where $\mathscr{L}$ is the projection into $[min(Freq), max(Freq)]$ segment, α, β are the SPSA step-size parameters.

The algorithm of our DVFS governor can be described as following:

1) Select initial value of an estimate $\hat{f}_0$.
2) Calculate Δ_n.
3) Perturb current optimal estimate $f_n = \mathscr{P}(\hat{f}_{n-1} + \beta\Delta_n)$.
4) Set current CPU frequency as f_n.
5) Obtain new noisy model observation $y_n = F(\hat{f}_{n-1} + \beta\Delta_n) + v_n$
6) Update frequency estimation $\hat{f}_n = \mathscr{L}(\hat{f}_{n-1} - \frac{\alpha}{\beta}\Delta_n y_n)$.
7) Go to Step 2.

Note that our model is parametrized with λ, α, β and γ, so the governor will express different behavior for different set of parameters.

B. Theoretical Foundation

To prove the consistency of the algorithm following by [13] we need to assume that changes of optimal frequency are bounded:

$$\|f_n - f_{n-1}\| \leq \delta < \infty.$$

Moreover, the investigated function $F(f)$ satisfies the following assumptions which are also required to prove the convergence of the algorithm from III-A.

Assumption 1: Function $F(f)$ is strongly convex and have a minimum point f^*:

$$\langle f - f^*, \nabla F(f) \rangle \geq \gamma \mathrm{ln}1.5 - 1, \forall \mathrm{f} \in \mathbb{R}.$$

Assumption 2:
The gradient $\nabla F(f)$ satisfies the Lipschitz condition:

$$\|\nabla F(f_1) - \nabla F(f_2)\| \leq \gamma 1.5^{\max(\mathrm{f}_1;\mathrm{f}_2)} \mathrm{ln}^2 1.5 \|\mathrm{f}_1 - \mathrm{f}_2\|,$$
$$\forall f_1, f_2 \in \mathbb{R}.$$

IV. Experimental Methodology

A. Device Selection

For our experiments we selected Xiaomi Redmi Note 8 Pro smartphone. It runs on Android 10, and its processor, Helio G90T has 2 clusters of cores, which are 2 A76 cores and 6 A55 cores [14]. It is important to note that from DVFS perspective cluster frequency can be controlled independently from one another.

A key consideration towards selecting this smartphone was an existing toolchain for running custom Android OS builds. We used begonia kernel (https://github.com/AgentFabulous/begonia) as a basis for our build and added SPSA implementation to the list of DVFS governors. Note that it was required to obtain root access for the smartphone to install custom build. Switching to SPSA governor was done by default `cpufreq` system calls [15].

We select OnDemand and Interactive DVFS governors among the available governors for the baseline of our experiments.

The source code for our DVFS governor is freely available (https://github.com/jackbogdanov/DVFS-for-begonia).

B. Test Cases

General-purpose DVFS governor should be able to handle various workloads. Each DVFS governor imposes a trade-off between energy consumption and performance, but also has a model of a workload it is designed to handle. As this model may or may not be suitable for real-world workloads, it is important to test DVFS governor using a diverse range of workloads. A following set of test cases was implemented:

1) Playing *Hill Climb Racing* game (https://play.google.com/store/apps/details?id=com.fingersoft.hillclimb).
2) Playing a song from *Spotify*.
3) Displaying and periodically scrolling PDF file in *Foxit PDF Editor* (https://play.google.com/store/apps/details?id=com.foxit.mobile.pdf.lite).
4) Watching *YouTube* video in 480p quality using default browser.
5) Watching *YouTube* video in 1080p quality using default browser.

All tests are launched 5 times for 15 minutes to average the influence of system background processes. They are implemented as Python scripts using `Monkeyrunner` test running tool (https://developer.android.com/studio/test/monkeyrunner). To launch a test, a smartphone needs to be connected to a controlling PC, and `Monkeyrunner` controls its execution through `adb` commands.

C. Device Preparation

We took the following measures in accordance to [16] to ensure the validity of our experiments and stability of its output:

- Uninstalling all unnecessary applications, turning off all application activities for those that couldn't be uninstalled.
- Turning off peripherals that were unused in a particular test case (i.e. Wi-Fi, 3G, GPS).
- We waited 2 minutes before beginning a new test in order to let background processes calm down and the device itself to cool down.
- A warm-up test execution was done before each set of test runs.

Note that as we're interested in CPU behavior there's no need to homogenize battery charge levels before each test run. Moreover, the device was connected via USB to a controlling PC, so the battery was charging during test runs.

D. Power Consumption Estimation

Android OS has special `time-in-state` files in the `/sys` directory that contain information about the time each processor core spent at a specific frequency. The data in these files is stored in pairs like "`<frequency><time>`". The number of pairs is equal to the number of frequencies a particular processor core supports. As different core clusters operate on different sets of frequencies, frequencies in `time-in-state` also differ from core to core depending on a cluster they belong to. Time is measured in 10 milliseconds units, and it is counted from the moment the corresponding driver was installed or reset to measure processor data. To accumulate timing information we reset statistics before each test run and gather it after a test run is finished.

In our experiments, we consider the CPU voltage to be constant, therefore to estimate power consumption we multiply timing data to corresponding weight coefficients in a device `power_profile.xml` - a file provided by smartphone manufacturer which contains power metrics for each smartphone device or peripheral. Its contents are shown in Table I. As the data there is stored in *mA* per cluster for 10 millisecond time intervals, our total power estimate is calculated in *mAh*.

TABLE I: Xiaomi Redmi Note 8 Pro CPU clusters frequencies

A55		A76	
Frequency (Hz)	Current (ma)	Frequency (Hz)	Current (ma)
2000000	90.04	2050000	324.33
1933000	85.8	1986000	307.98
1866000	80.27	1923000	291.52
1800000	72.77	1860000	269.61
1733000	66.61	1796000	247.53
1666000	62.05	1733000	233.56
1618000	58.95	1670000	209.73
1500000	52.33	1530000	177.39
1375000	44.83	1419000	152.46
1275000	39.69	1308000	130.33
1175000	35.5	1169000	105.19
1075000	31.24	1085000	91.11
975000	27.86	1002000	79.53
875000	25	919000	70.65
774000	23.5	835000	61.38
500000	19.55	774000	56.85

V. Experiments

Our experimental results are openly available[1], and Table II contains total energy consumption values translated to *mAh*.

We found that Hill Climb Racing test proved to be the worst case for OnDemand governor - it consistently turns both A55 and A76 cores to maximum frequency. SPSA runs slightly better than Interactive, because it keeps A55 at lower frequencies, although its A76 frequencies distribution is wider.

As seen in Fig. 1 and 2, Spotify test is where governors behave noticeably differently. Interactive overestimates the need for maximum performance of A76 cores, and SPSA uses all available frequencies for A55, while OnDemand keeps them closer to lower boundary.

Although Foxit PDF Editor test is not a power consuming one at first glance, rendering new page of a PDF file is resource intensive. SPSA wins there as it doesn't overuse A76 cores.

Interestingly, power distribution difference between Youtube 480p and YouTube 1080p tests is marginal despite considerably larger amount of data being transferred in second test. Interactive tends to overuse maximum frequency of A76 cores, thus being most power-hungry governor. OnDemand is more conservative both in A55 and A76 cores than SPSA therefore consuming less energy.

While the energy savings are important, it's also important that they do not result in a laggy performance for the end-user. With this in mind we compared performance of SPSA governor using `AnTuTu` performance benchmark (https://www.antutu.com/en/index.htm). This is a commonly used benchmark among Android developers to assess the performance of various smartphones, and the values it provides show an integral performance metric for the smartphone.

[1]https://drive.google.com/file/d/17ki2HxrFVwYjccV4-pPwIZuD8g8iqoLG/view

TABLE II: Energy consumption of SPSA, OnDemand and Interactive governors

Test	SPSA (in mAh)	OnDemand (in mAh)	Interactive (in mAh)
Hill Climb Racing	34.196	91.862	37.565
Spotify test	33.913	24.678	41.343
Foxit PDF Editor	54.729	65.322	77.950
YouTube 480p	32.957	24.579	47.980
YouTube 1080p	32.378	25.319	47.254

TABLE III: AnTuTu benchmark scores

Governor	CPU	GPU	Memory	UX	Total
Powersave	28883	50193	30792	18458	128326
SPSA	80555	74357	40897	48442	244251
OnDemand	83704	76335	42020	49893	251972
Interactive	79459	76880	39403	49412	245154
Performance	86756	79952	42364	51789	260861

The benchmark runs several tests grouped by four categories: CPU, GPU, Memory and User Experience (UX). Table III presents the scores of SPSA, OnDemand and Interactive in all four categories and the Total column shows their sum as an integral quality metric. Powersave and Performance governors' results are shown as a reference minimum and maximum values.

The performance difference between SPSA, OnDemand and Interactive is close to be visually insensible by the end user (3.06% and 0.36% SPSA difference with OnDemand and Interactive governors respectively), and it is comparable with the best performance available (6.38% difference with Performance governor). However looking at the GPU, memory and user experience, there's a trend that DVFS governor affects the performance of other smartphone peripherals and therefore has a system-wide performance impact with worse performing governors obtaining consistently lower scores in each area.

VI. Conclusion and Future Work

The results of our SPSA application to modern smartphone DVFS in current article along with our previous work [17] show that it's a good all-round algorithm performing at the level of standard DVFS algorithms and sometimes noticeably outperforming them. The data shows that SPSA governor under current parameters is slower to build up CPU frequency, but also slower to cool down than OnDemand and Interactive. Therefore in a scenarios where workload is evenly distributed over time or has prolonged periods of calculations SPSA performs better than OnDemand or Interactive. In a spiky workloads OnDemand and especially Interactive are quick to react, but also quick to turn off excessive CPU frequency, while SPSA interprets these spikes as a constant need for CPU resources. Overall, we consider current SPSA governor to be viable for everyday usage.

Investigating how different parameters affect viability of SPSA governor for different usage scenarios is a straightforward next step in its development. Another interesting research direction appeared from the observations of behavioral differences between OnDemand, Interactive and SPSA. As there are standard DVFS governors like OnDemand and Interactive on virtually all modern smartphones, and some tasks in applications are cyclical in nature (like data refreshes and updates, periodical update downloads etc.), to our knowledge there wasn't yet found a relation between task frequency, supported CPU frequencies, DVFS algorithm and energy consumption. As common users are highly unlikely to change DVFS governor if even know they exist, finding such a relation might help software developers to create energy profiles for their applications adapting their task timers to save energy based on specific workload expectations from a current DVFS governor.

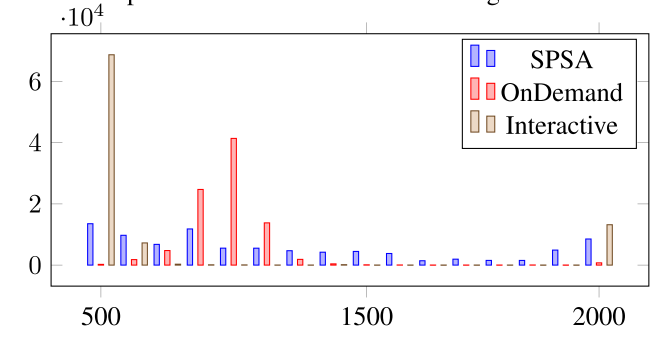

Fig. 1: Spotify test A55 cluster

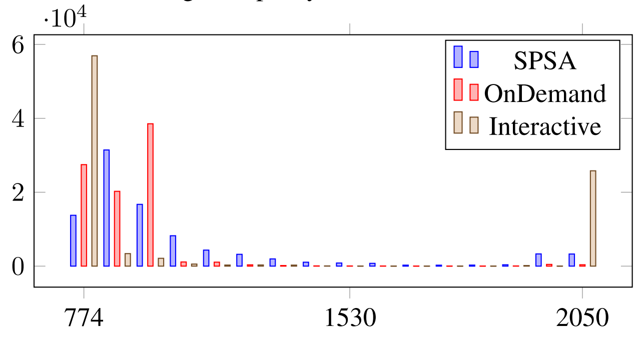

Fig. 2: Spotify test A76 cluster

References

[1] C. Sahin, F. Cayci, I. Manotas, J. Clause, F. Kiamilev, L. Pollock, and K. Winbladh, "Initial explorations on design pattern energy usage," *2012 1st International Workshop on Green and Sustainable Software, GREENS 2012 - Proceedings*, 06 2012.

[2] D. Brodowski and N. Golde, "Cpu frequency and voltage scaling code in the linux (tm) kernel. linux cpufreq. cpufreq governors," https://android.googlesource.com/kernel/msm/+/android-7.1.0_r0.2/Documentation/cpu-freq/governors.txt, 2015.

[3] J. C. Spall, "Multivariate stochastic approximation using a simultaneous perturbation gradient approximation," *IEEE Transactions on Automatic Control*, vol. 37, no. 3, pp. 332–341, 1992.

[4] O. Granichin and N. Amelina, "Simultaneous perturbation stochastic approximation for tracking under unknown but bounded disturbances," *IEEE Transactions on Automatic Control*, vol. 60, no. 6, pp. 1653–1658, 2015.

[5] O. Granichin, L. Gurevich, and A. Vakhitov, "Discrete-time minimum tracking based on stochastic approximation algorithm with randomized differences," in *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*. IEEE, 2009, pp. 5763–5767.

[6] Y. L. Chen, M. F. Chang, C. W. Yu, X. Z. Chen, and W. Y. Liang, "Learning-directed dynamic voltage and frequency scaling scheme with adjustable performance for single-core and multi-core embedded and mobile systems," *Sensors*, vol. 12, no. 9, Sep 2018.

[7] J. Lee, S. Nam, and S. Park, "Energy-efficient control of mobile processors based on long short-term memory," *IEEE Access*, vol. 7, pp. 80 552–80 560, 2019.

[8] A. Das, M. J. Walker, A. Hansson, B. M. Al-Hashimi, and G. V. Merrett, "Hardware-software interaction for run-time power optimization: A case study of embedded Linux on multicore smartphones," in *Proceedings of the International Symposium on Low Power Electronics and Design*, Jul 2015.

[9] X. Li, W. Wen, and X. Wang, "Usage history-directed power management for smartphones," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Dec 2015.

[10] K. Poornambigai, M. L. Raj, and P. Meena, "Reducing the energy consumption using dvfs performance optimizing scheme," *EPRA International Journal of Research and Development (IJRD)*, vol. 2, no. 1, Jan 2017.

[11] L. Broyde, K. Nixon, X. Chen, H. Li, and Y. Chen, "MobiCore: An adaptive hybrid approach for power-efficient CPU management on Android devices," in *2017 30th IEEE International System-on-Chip Conference (SOCC)*, Sep 2017, pp. 221–226.

[12] G. O. N., "Linear regression and filtering under nonstandard assumptions (arbitrary noise)," *IEEE Transactions on Automatic Control*, vol. 49, no. 10, pp. 1830–1837, 2004.

[13] O. Granichin and A. Vakhitov, "Accuracy for the spsa algorithm with two measurements," *WSEAS Transactions on Systems*, vol. 5, 05 2006.

[14] "Xiamoi Redmi Note 8 pro specifications," https://www.mi.com/global/redmi-note-8-pro/specs/, 2020, [Online; accessed 19-March-2021].

[15] "Android OS cpufreq specification," https://android.googlesource.com/kernel/common/+/a7827a2a60218b25f222b54f77ed38f57aebe08b/Documentation/cpu-freq/index.txt, [Online; accessed 19-March-2021].

[16] V. Myasnikov, S. Sartasov, I. Slesarev, and P. Gessen, "Energy consumption measurement frameworks for android os: A systematic literature review," in *Proceedings of the Fifth Conference on Software Engineering and Information Management 2020 (SEIM 2020)*, ser. CEUR Workshop Proceedings, 2020. [Online]. Available: http://ceur-ws.org/Vol-2691/paper10.pdf

[17] E. Bogdanov, A. Bozhnyuk, S. Sartasov, and O. Granichin, "On application of simultaneous perturbation stochastic approximation for dynamic voltage-frequency scaling in android os," in *7th International Conference on Event-Based Control, Communication and Signal Processing (EBCCSP'21)*, 2021.