Computational Cost Analysis and Data-Driven Predictive Modeling of Cloud-based Online NILM Algorithm

Computational Cost Analysis and Data-Driven Predictive Modeling of Cloud-based Online-NILM Algorithm

Mulugeta Weldezgina Asres, Luca Ardito, *Member, IEEE* and Edoardo Patti, *Member, IEEE*

***Abstract*—Online non-intrusive load monitoring algorithms have captivated academia and industries as parsimonious solutions for household energy efficiency monitoring as well as a safety control, anomaly detection, and demand-side management. However, the computational energy cost for executing such algorithms should not overcome the promised energy efficiency from the disaggregated appliance specific consumption information feed-backs. Moreover, the energy efficiency of cloud computing systems is also becoming a concern for the environment due to carbon emission. This study analyzes the energy spent to execute NILM algorithms via computation cost estimation and prediction using computing system-level power monitoring and data-driven approaches. A generic framework for an automated algorithm cost monitoring and modeling methodologies is devised for large load scale deployment of Cloud-based Online-NILM algorithms. The efficacy of the proposed approach was examined and validated on two computing system use-cases, i.e., Dedicated Server and Cloud Virtual Server. The prediction models, developed using statistical and machine learning tools, demonstrate the promising applicability of the data-driven approach with a very high prediction accuracy without detailed knowledge of the computing systems and the algorithm.**

***Index Terms*—NILM, Load Disaggregation, Data-Driven, Computational Cost, Algorithm, Machine Learning, Cloud Systems.**

ACRONYMS

APE Absolute Percentage Error. 12
CI Confidence Interval of APE. 12
$MAPE$ Mean Absolute Percentage Error. 12, 13
$MaxAPE$ Maximum APE. 12, 13
N_h Number of houses. 8, 12
N_hs List of number of houses. 5, 8
N_p Number of parallel processors. 8, 10, 11
N_ps List of number of parallel processes. 5
$N_{h\_capacity}$ Maximum N_h can be processed in dw. 12
PE Processing Energy Cost. 2, 10–12
PT Processing Time Cost. 2, 8, 10–12
PT_{code} In code processing time measurement. 5, 10
$R-value$ Regression Fit Score. 12, 13
$RMSE$ Root Mean Square Error. 12, 13
dw Online disaggregation time-window. 11, 12
puh per-unit house. 8, 10
$syncLen$ Synchronization pattern length. 7
HAGP House AGgregate Power Loads. 4
Large HAGP *HAGP* for a given large N_h. 4, 5
Unique HAGP Unique online one-hour HAGP loads. 4, 5

CbON Cloud-based Online-NILM Algorithm. 2, 4–8, 12, 13
CVS Cloud Virtual Server. 1, 2, 8–13

DS Dedicated Server. 1, 2, 8–13

FHMM Factorial HMM. 4, 13

HMM Hidden Markov Model. 4

ME Mean Estimation. 8, 12, 13

NILM Non-intrusive Load Monitoring. 1–13

RF Random Forest. 8, 12, 13
ROI Region of Interest. 6–10

WUM WattsUp? Pro Meter. 2, 3, 5–13

M.W. Asres, L. Ardito and E. Patti are with the Dept. of Control and Computer Engineering, Politecnico di Torino, Turin, Italy (e-mail: {name.surname}@polito.it).

I. INTRODUCTION

THE parsimonious non-intrusive approaches have attracted academia and industries as viable solutions for household energy efficiency monitoring. These techniques are collectively known as Non-intrusive Load Monitoring (NILM).

The concept was introduced by Hart [1] as a method for extracting electrical loads by examining their power consumption signatures within the aggregated load data. The non-intrusiveness lower complexity during implementation and maintenance, which immensely reduces incurring costs while leveraging the convenience of usage as compared to intrusive sub-metering approaches [2]. Furthermore, the applications of NILM are suggested beyond energy efficiency into safety control, anomaly detection, and demand-side management.

Generally, literature dictates consumers' energy efficiency can be enhanced by up-to 12% of annual power consumption with valuable appliance-specific consumption feed-backs [3], [4]. However, some intriguing questions draw attention: How much is the computational energy cost for running NILM systems? Does the energy spent to execute the NILM algorithm have an impact on the expected energy efficiency obtained from using the disaggregated load information? Can these energy costs be modeled using data-driven methods for prediction to meet scaling requirements? This study is presented to answer these questions and propose generic data-driven computational cost analysis and modeling approaches for executing a NILM algorithm. The approaches are aimed

at providing relevant insights that aid decisions for effective deployment strategies of large scale NILM considering computational energy and processing time costs.

Software or algorithm energy cost can be measured or estimated using various techniques [5]–[18] such as hardware-based (computing system power measurements or embedded power-sensors on motherboards), model-based (resource usage models) or software-based (kernel APIs (Application Programming Interfaces)). A non-intrusive system-level power measurement technique [5], [6] is adopted in this study due to its deployment convenience (unlike intrusive embedded power-sensors), lower requirement for detail knowledge of computing system (unlike hardware or operating system specific model-based and Kernel-APIs tools), and general applicability for servers (unlike platform-dependent software-based methods). Hence, we employ WattsUp? Pro Meter (WUM) [19] to monitor the power consumption of the system running the NILM algorithm.

The main challenge of the hardware-based measurement approach is the inability for evolution and the difficulty to scale despite their high precision in energy measurement at a high level of granularity [18]. This limitation is owing to intrusiveness (for embedded power chips in the motherboard) or inferring the power consumption of the monitored algorithm from the dynamic power of the computing system (for external power meters such as WUM). Hence, we proposed an automated and effective cost extraction technique based on synchronization pattern insertion [17] to ensure a scaleable monitoring system.

Organizations are currently focused on attaining promising technologies using Cloud systems to reduce their carbon impact while curtailing operational costs [20]. Hence, we proposed our cost monitoring system for Cloud-based Online-NILM Algorithm (CbON), which provides real-time load disaggregation but runs on Cloud premises. In our experiment, we have monitored, analyzed and modeled Processing Time Cost (PT) besides Processing Energy Cost (PE) as computational costs of a use-case CbON algorithm from [21]. The estimated costs of the algorithm are extracted from the WUM power monitoring data using our proposed approaches. The study was carried out considering a large-scale NILM deployment scenarios, such as a large number of house loads and parallel processing capabilities.

Finally, our analytical results on two different use-case computing systems, i.e., Dedicated Server and Cloud Virtual Server, demonstrate computational costs of a NILM algorithm can be monitored and modeled effectively with very high accuracy via data-driven approaches from system power measurements. The efficacy is found to be consistent and robust that further validates the promising applicability of the data-driven approach without detailed knowledge of the underlying computing system.

Our study is focused on CbON algorithms in particular, yet most of the proposed data-driven methods can be easily adapted for monitoring Cloud-based applications, other than NILM. Although energy efficiency studies for Cloud systems mainly focus on optimizing the energy via task scheduling algorithms [20], virtualization technologies [14] and load-balancing across different virtual machines or servers [22], measuring the energy consumed by a running application can be remarkably beneficial for Cloud system. Even a miniature amount of saving per Virtual Machine can add up to a substantial gain in large-scale Cloud systems [13], [15], [20].

The key contributions of this study are briefed below:

- We present a study on the data-driven computation costs monitoring, analysis, and predictive modeling of a Cloud-based Online-NILM Algorithm.
- To the best of our knowledge, our work pioneers a study in assessing the energy cost of a NILM algorithm.
- We develop a generic end-to-end computational cost monitoring and modeling methodologies from computing system power measurement without detailed knowledge of the underlying system and the NILM algorithm.

The rest of this paper is organised as follows. The background concepts of the study are briefly discussed in Section II. Section III describes the data-set, while Section IV presents the methodology. Section V discusses the experimental results. Finally, Section VII provides our concluding remarks while threats to validation are highlighted in Section VI.

II. Background

This section discusses the background concepts and related works of the study.

A. Software Energy Measuring and Modeling

Noureddine et al. [18] review different energy measurement approaches that can be classified as measurement/estimation and modeling. In the first sub-category, the goal is to determine the energy consumption through the hardware equipment, while the latter creates a mathematical model of the software energy consumption to provide energy data without external equipment. Hindle et al. [23] proposed an approach to measure how the energy consumption of software applications varies through the different versions.

There are three ways, i.e., instant power measurement, time measurement, and model estimation, for getting power/energy consumption of software from a device [17]. Table I analyses the benefits and drawbacks of each technique.

a) Instant Power Measurement: this technique measures the instantaneous power consumed by a device [5], [6], [11], [24]. The integral of the power values over a period of time gives energy. However, the instant power measurements are precise if the sampling frequency is high. Although the approach usually operates at the device level, hardware component-level measurement is possible.

b) Time Measurement: estimate the energy consumption of a device through measurement of time [25], [26]. If we assume constant consumption over time, the speed at which energy is consumed strictly depends on the device's power consumption. For this reason, the average power consumed can be retrieved by measuring the time needed to completely discharge a battery. The more the battery capacity is precise, the more the measurement is reliable. The drawback of this method is that it is challenging to employ it for data centers because it would require large batteries, and the measurement

Measurement Technique	Pros	Cons
Instant Power Measurement	Precise if the sampling frequency is high	Physical instrumentation needed. Difficult to isolate a single software application's contribution
Time Measurement	Precise if the exact energy stored in the battery is known	Requires many repetitions of long tasks. Difficult to isolate a single software application's contribution
Model Estimation	No instrumentation required. Easy to isolate a single software application's contribution	Precision is not always declared

TABLE I: Evaluation of measurement techniques

would be imprecise one in terms of granularity (i.e., it would be almost impossible to isolate the execution of a given software program).

c) Model Estimation: consumption measurements through models are calculated by relating the power consumption of a particular device with its internal peripheral usage indicators, such as the CPU states, instructions, memory or disk accesses, and network adapters. For example, [10], [13]–[15] discussed software-based method for energy consumption estimation in Cloud and virtualized environment from CPU system-logs using platform dependent Kernel APIs[1,2]. Similarly, [27]–[29] introduced an approach for building a power model for Android devices by using APIs to retrieve a variety of states, including the battery, network connection, Wi-Fi, and screen.

Model estimation requires choosing a model suitable for the device on which the software will run and also has overhead caused when extracting the resource utilization. The overhead is a critical value because a software process implementing the model executes the resource usage data collection that affects the consumption of the device on which it is executed. Most of the works, proposed to build energy models based on resource utilization, require a detailed knowledge on the working principles and states of resources such as CPU [5]–[11], Memory [5], [12] and Disk [5], [16].

Generally, previous efforts for energy cost estimation of a running application on data-center or Cloud servers use energy/power monitoring hardware device or kernel APIs [10], [13]–[15], [30]. The hardware-based methods either employ power meters to measure the server instant power directly [5], [6], [11] or connect power sensors into motherboard [24]. Further detail of the summarized discussion on modeling approaches from various literature can be found in [5]. Overall, the main challenges of the methods are related to the overhead caused by reading the resource usage information (model-based), intrusive techniques are not widely applicable due to inconvenience of installation (sensor-chip on the motherboard), and limited to certain operating systems or vendors (kernel APIs), and background noise from other processes and need synchronization along with efficient active region extraction (computing system-level power).

For our use-case in server systems, we employed our own customized system-level instant power measurement techniques using external power monitoring devices. The approach was chosen as it can be applied without having detailed knowledge of the underlying computing system or server. Moreover, we attempt to mitigate the challenges of system-level instant power measurement, such as interference from other running applications besides the target software and the requirement of a high sampling rate. Furthermore, to the best of our knowledge, there is no published study on computational energy cost analysis and modeling for NILM algorithm.

[1]Linux Powertop: https://01.org/powertop/

[2]Joulemeter: https://www.microsoft.com/en-us/research/project/joulemeter-computational-energy-measurement-and-optimization/

B. External Power Monitoring Devices

External power recording devices such as WUM [5] and Plogg [6] have been employed in many studies [5], [6], [19], [30]–[32], [32], [33]. Their data measurements can be retrieved via serial, Ethernet, or even Bluetooth connections. External meters are easy to deploy on any device fed through a power plug; this simplicity makes them a good alternative to embedded sensors plugin motherboards. Results from [6] showed that the external meters could also provide more accurate measurements than some embedded sensors.

In our study, we used WUM to monitor the power consumption of servers. WUM measures the power/energy consumption of appliances and devices that plug into electrical outlets for their power supply [19]. The device can monitor a wide variety of real-time electricity usage data, including voltage, current power, and power-factor. It was utilized as an efficient means of collecting electricity energy reading in various studies [19], [31], [32] including popular public NILM data-sets such as UK-DALE [33]. Furthermore, some studies [5], [30], [32] have adopted it to validate alternative software energy estimation approaches. Further evaluation of the capability and limitations of the meter can be found in [19]. We provide specifications of the WUM, which was employed in our study, in Table II.

Parameters	Characteristics
Measurement Accuracy	+/- 3% (loads above 10 watts), +/-5% (loads below 10 watts)
Power supply	100-277 Volts AC, 50/60 Hz, 20 amps
Supply voltage fluctuations	Not to exceed +/- 10% of the nominal voltage

TABLE II: WUM specification

C. Non-intrusive Load Monitoring Algorithms

NILM algorithms can be categorized as Offline or Online systems. Online-NILM methods employ time slice or

window-based methods for real-time detection and learning of appliance consumption [21], [34]–[40]. On the other hand, Offline-NILM does not perform real-time load monitoring and typically requires daily readings to execute load disaggregation [1], [41]. However, the merits of instant load information have been acknowledged for unlocking new grid services such as demand-side management, which raises interactivity in energy awareness, leading to more green behaviors [21], [34]–[36]. Real-time information is also very crucial for safety control, such as preventing a fire from forgotten switched on heating devices. Online-NILM can be deployed at the measurement edges in smart meters or remotely in Cloud servers [21], [40]. Cloud-based Online-NILM Algorithm aims to enable online load detection while reducing the hardware and software complexity of running Online-NILM algorithms on edge devices, i.e., smart energy meters.

Most of the NILM state-of-the-art approaches employ Hidden Markov Model (HMM) and its variants to provide real-time online load disaggregation [21], [35]–[40]. Hence, our previously published Cloud-based Online-NILM Algorithm using HMM [21], hereafter referred as CbON, was employed as use-case for our experiments for the computational cost of NILM algorithms. Our choice for the algorithm is mainly driven by the relevance of Cloud-based deployment for NILM, as previously explained, as well as the better efficacy than other state-of-the-art NILM algorithms with public benchmark data-sets (see [21] for details). The algorithm combines the strength of an event-based approach for unsupervised model development and accurate disaggregation from the supervised HMM method. Moreover, we had full access to the code implementation, and the algorithm was deployed in a Cloud smart metering framework [42].

The NILM algorithm, from [21], was proposed to address the problem of providing fast and online household appliance load detection. It is composed of two methodologies: i) Unsupervised event-based profiling and ii) Markov chain appliance load modeling. The event-based part performs event detection through contiguous and transient data segments, events clustering, and matching. The resulting features are used to build household-specific HMM appliance models. The load disaggregation is then performed online using an Additive Factorial HMM (FHMM) from the generated appliance HMM models for a given aggregate power reading.

Generally, the working flow of the NILM system includes sending hourly power reading segments from the smart meter to a Cloud SQL database, and then the CbON algorithm retrieves the ingested power data and performs load disaggregation. Furthermore, the CbON comprises loading settings, appliance models, and aggregate power consumption during the given disaggregation window from the database, performs load disaggregation, and stores results back into the SQL database. In our cost study, only the computational of the CbON Algorithm tasks were considered. The CbON has three sub-modules, i.e., Pre-processing, Dynamic FHMM for Aggregate Consumption Modeling, and Disaggregation. In the Pre-processing, a median filter is applied to smooth the signal by removing spikes and outliers followed by local-background power estimation. Embarking with the estimation of the likelihood of appliance activation based on load features, FHMM generates an aggregate power model by combining the appliance HMMs. The appliance states and power emissions from the aggregate model are decoded in the final Disaggregation stage. Nevertheless, since our proposed approach is data-driven, a prior detailed knowledge of the NILM algorithm is not required, and it is treated as a block-box in our cost monitoring system.

III. Data-set Description and Preparation

For our study, we prepared two sets of data, i.e., i) large scale house energy data-set and ii) monitored computation cost data-set.

We generate the large scale house energy data-set to provide computation evaluation at a large scale deployment of the CbON algorithm. It was developed from some of the widely used public NILM data-sets with real house energy profiles, i.e., REDD [43] and UK-DALE [33]. The public data-sets were also employed in the reference CbON [21] algorithm for appliance HMM model building. The REDD data-set contains low-frequency power data for six US houses, while the UK-DALE data-set comprises five UK houses. Seventy-eight days of household data were prepared from a period of around two weeks of data from each of these public data-sets (see Table III). The daily load data were organized after data cleaning and focusing on the previously built appliance models in the target CbON from [21].

Generally, the large scale simulated house aggregate energy data-set generating algorithm is composed of data chunking, shuffling, and duplication techniques to synthesize power demands of large number of houses (see Figure 1). The real data from REDD and UK-DALE are organized in daily format to form a House AGgregate Power Loads (*HAGP*) data baseline *(a)*. Then each daily data (24 hours) is segregated in hourly fashion using the disaggregation window ($dw = 60minutes$) to generate 1872 Unique online one-hour HAGP loads (*Unique HAGP*) data-set *(b)*. To generate *HAGP* for a given large N_h, i.e. *Large HAGP* data-set, the *Unique HAGP* data-set is randomly shuffled to introduce entropy and duplicated N_d times in (f), where N_d is a duplication factor which is calculated as N_h divided by the size of *Unique HAGP* in (e).

Both *Unique HAGP* and *Large HAGP* data-sets contain only the house identification, $houseId$, and the start and end timestamps of the online data window, $dw\_timestamps$, to have a fast and memory-efficient data generation while keeping similar process flow as the load disaggregation algorithm. The actual load data is only retrieved from the database using $houseId$, and $dw\_timestamps$ information during load disaggregation when the NILM algorithm is executed. Although the CbON algorithm was designed to work with 15, 30, 45, and 60 minutes online data [21], a one-hour window was used in our experiments due to relatively better disaggregation accuracy, and lower online data transfer overheads. Moreover, users less frequently change appliances operation states in the smaller windows in a typical residential load.

The computation cost data-set was prepared for the cost analysis and modeling by monitoring the processing time and

Public Data-set	House IDs	Total Number of Days
REDD	1-4, 6	49
UK-DALE	9001 - 9005	29

TABLE III: Number of selected days from public data-sets

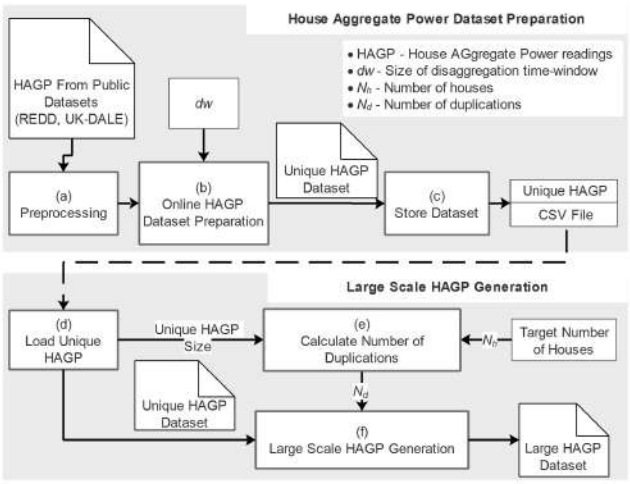


Fig. 1: *Large HAGP* data-set generation

energy consumption of the NILM algorithm in dedicated and cloud virtual servers in various scenarios such as variation in the number of houses and parallel processing capabilities.

In our experimental setup, the number of multiprocessing processes was varied from 1 to 8 (maximum number of processors in the system) while the number of houses ranges from 1000 to 2500 with 500 step and large scale scenarios from 4000 to 10000 with 2000 step. The experiment generated 640 data points for 64 unique scenarios.

IV. Methodology

The proposed methods for computation cost measuring, analysis, and predictive modeling of the CbON algorithm are discussed in this section. In this study, computational costs of the algorithm when disaggregating one-hour online load reading are analyzed and modeled in terms of the execution time and energy consumption. The trade-off of multiprocessing scenarios (generally, increasing power while reducing processing time with additional processing cores) is also incorporated to provide a comprehensive study on the computational costs. A single execution of the CbON algorithm is composed of setting and configuration loading, time-window load data and appliance HMM models retrieval for a given house from SQL database, load disaggregation (the core section of the algorithm), and finally, storing disaggregated appliance level load data back into the SQL.

A. Flow Diagram of the Framework

A general flow diagram of the proposed cost monitoring, analysis, and modeling framework is illustrated in Figure 2. The diagram comprises generating the synthesized power reading from a large number of houses (discussed in Section III), measuring system instant dynamic power consumption using WUM and extracting of the NILM algorithm costs, and finally, cost analysis and prediction model development. The major blocks of the framework are briefly discussed below.

a) External Inputs: comprises external inputs to the cost monitoring such as the number of houses selections, List of number of houses (N_hs), *Unique HAGP* data-set and CbON algorithm, and load data from REDD and UK-DALE in database.

b) WUM Interface: consists of tasks involving interaction with the WUM. In the beginning, the power supply is connected to the computing system or server through WUM in between. The WUM also connects to the server via USB connection for configuration management and data acquisition. Then, WUM settings such as *data storage location: meter's internal memory, overwrite during memory full: false, sampling time: 1 second and parameters to record: power in Watts* are configured. Just before the cost measurement starts, the memory of the WUM needs to be cleared manually, and the meter automatically starts recording the server's power consumption. After the power monitoring finishes, *WUM power log* from memory of the meter are exported into an external file, *WUM log CSV file*, for later use.

c) Cost Monitoring: Before starting the system power monitoring, the inputs variables such as *Unique HAGP* data-set and CbON algorithm from the *External Inputs* block are loaded along with load scales N_hs and multiprocessing N_ps settings. Based on the settings, the target CbON algorithm is executed while the server system power is being monitored by the WUM simultaneously. In code processing time measurement using *Python Time* library, is also employed to further validate the cost monitoring using WUM.

d) Cost Analysis and Modeling: following the completion of the cost monitoring, *WUM log CSV file* and the PT_{code} are loaded to develop the (*Cost data-set*). The cost data-set is then used for analysis and modeling of the computational costs of the CbON algorithm.

B. Multiprocessing

At the operating system level, multiprocessing refers to the execution of multiple concurrent processes in a system with each process running on separate cores. Though the term multiprocessing may signify different meanings in different contexts, in our scenario, it implies parallel execution of multiple processes using more than one processor. In our experiments, the CbON algorithm runs on a single process when disaggregating a single online data segment per house.

We employ *Asynchronous Pool* Python API to parallelize execution of the CbON algorithm on input load data from multiple houses, i.e., data parallelism. Depending on the platform, multiprocessing can use three ways to start a process, such as a spawn, fork, and forkserver. However, only spawn is currently supported on our windows based system. In the spawn mechanism, a parent process starts a fresh Python interpreter process, and a child process will only inherit those resources necessary to run, the process objects excluding unnecessary handles from the parent process [44]. Hence, spawning was employed to start the load disaggregation

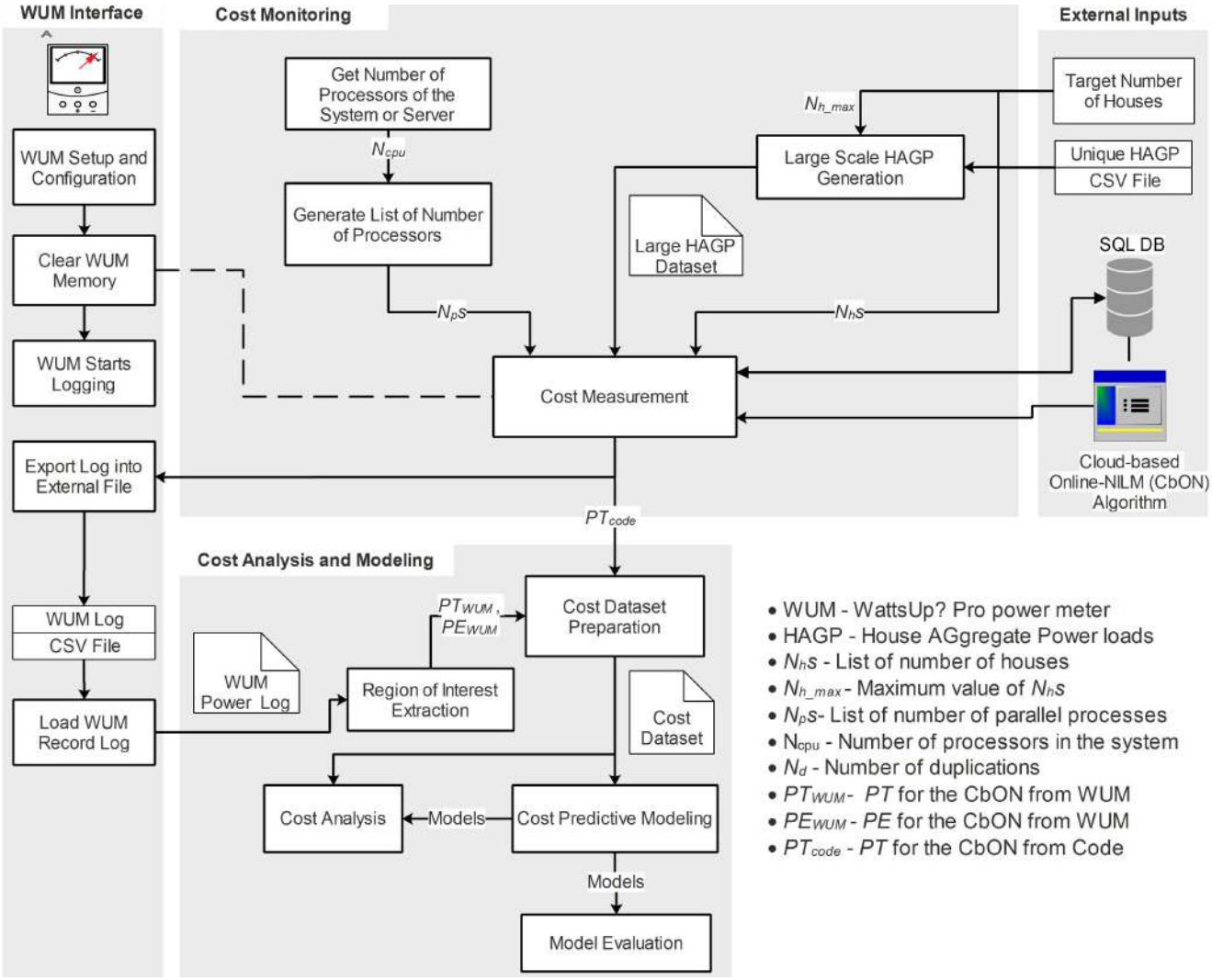


Fig. 2: Computational cost monitoring, analysis and modeling methodology for CbON algorithm

process during multiprocessing in our experiment. The list of the possible number of processes for multiprocessing is $N_p : 1, 2, ..., N_{p\_max}$ where N_{p_max} is the number of logical processors in the system. Generally, multiprocessing beyond N_{p_max} is not recommended due to the limited computation leverage from high context switching [44] though it is possible.

C. Computational Cost Monitoring

Before starting the system power consumption monitoring, we let the computing system stabilize by waiting to finish or closing active processes until only background processes that do not cause significant power fluctuation on the system remain. This prevents unexpected processes from corrupting the measurement when server power consumption is monitored by WUM.

To effectively use WUM for algorithm energy consumption, a method to synchronize or correlate the start and end times of the target process within the WUM record is mandatory. Thus, to determine the Region of Interest (ROI) corresponds to the NILM process execution, a synchronization marker insertion (*AddSyncMarker*) was proposed in [17] inspired by signal synchronization techniques in digital communication systems. The technique involves sleep-wake-sleep mode to generate a kind of square wave in the system power reading (see Figure 3). The synchronization pattern insertion function block is called before and after the target algorithm code execution (see detail in Section IV-D). The square wave patterns provide marking waveform, which can be easily spotted manually for simple experiments or using pattern matching algorithms such as Dynamic Time Wrapping [45] for large scale experiments. The technique was found to be effective in locating the power signal of the target process from the system-level power waveform. However, due to signal corruption from background processes (e.g., in virtual machine servers), the square-wave can be distorted, limiting the efficacy of pattern matching algorithms. Hence, we propose a new synchronization pattern (see Algorithm 1) that it is easily locatable with simple signal processing using trend analysis and clustering techniques.

When multiple recursive scenarios are included in a single cost monitoring experiment, *AddSyncMarker* is executed at the beginning and end of each case. Hence, the starting and ending section of an outer scenario will have one more synchronization pattern than the inner ones. This allows us to locate the region of interest in the WUM reading uniquely

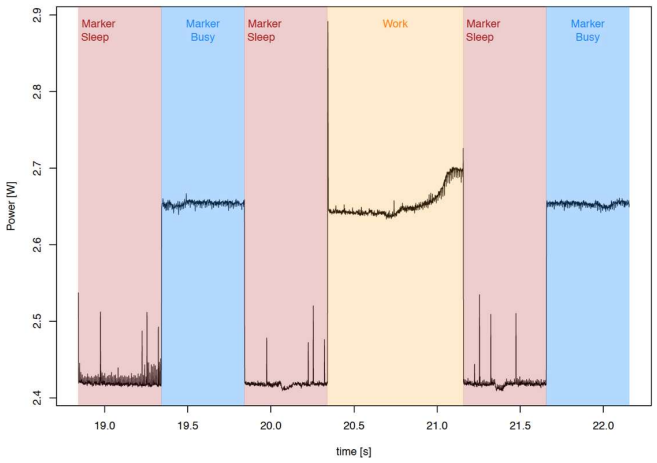


Fig. 3: Sample WUM synchronization marker illustration [17]

Algorithm 1 Synchronization Marker Insertion

```
1:  procedure AddSyncMarker(syncLen)
    ▷ syncLen: Synchronization pattern length
    ▷ syncLen in seconds
2:      Low(syncLen)
3:      High(syncLen)
4:      Low(syncLen)
5:      procedure High(syncLen)
6:          timeout ← CurrentTime() + syncLen
7:          while True do:
8:              if CurrentTime() > timeout : then
9:                  break
10:     procedure Low(syncLen)
11:         Sleep(2 × syncLen)
```

using the added length of the consecutive patterns. Our study, has four different scenarios from top to lower levels, i.e., start and end of the entire cost monitoring, experiment repetitions, and the basic scenarios from different load scales and multiprocessing. Accordingly, the lowest scenario marks the different number of parallel processes selections with single synchronization patterns with length of $5 \times syncLen$ ($syncLen$ is duration the awake section of the pattern, see Algorithm 1), while the different number of houses selection scenarios, experiments repetitions, and all the cost analysis experiments are marked with $10 \times syncLen$, $15 \times syncLen$, and $20 \times syncLen$ respectively. Experiment repetitions are incorporated to normalize noisy measurements, especially as some unexpected background processes can corrupt the WUM measurement.

D. Region of Interest Extraction from WUM Reading

After system-level power monitoring, the activity regions which belong to the CbON algorithm need to be extracted from the timer-series power logs of the WUM using the synchronization patterns. The proposed approach for extraction of the active time-window section corresponding to the algorithm under monitoring from the background power consumption and isolation of ROI for the different experiment scenarios is discussed in this section.

The extraction embarks by spotting the synchronization pattern signals, using a pattern extraction algorithm, from the WUM power reading. The extraction algorithm employs

Algorithm 2 Computational Cost Measuring

```
1:  procedure CostMeasurement(dw, genHouseIds, N_h s, Nes)
2:      N_p_max ← GetSystemNumberofProcessors()
3:      N_p s ← range(start = 1, end = N_p_max, step = 1)
4:      expIds ← range(start = 1, end = Nes, step = 1)
5:      syncLen ← 4                                   ▷ in seconds
6:      SaveExpSetting(dw, Nes, N_h s, N_p s, syncLen)
7:      AddSyncMarker(syncLen)
8:      procCostTable ← []
9:      for expId in expIds : do
10:         AddSyncMarker(syncLen)
11:         for N_h in N_h s : do
12:             AddSyncMarker(syncLen)
13:             houseIds ← GetRandomizedHouseIds(genHouseIds, N_h)
14:             for N_p in N_p s : do
15:                 AddSyncMarker(syncLen)
16:                 t_start ← GetTimer()
17:                 procCost ← MultiProcessingNILM(N_p, houseIds)
18:                 t_end ← GetTimer()
19:                 procTime ← t_end − t_start
20:                 procCostTable ← join(expId, N_p, N_h, procTime)
21:                 AddSyncMarker(syncLen)
22:             AddSyncMarker(syncLen)
23:         AddSyncMarker(syncLen)
24:     AddSyncMarker(syncLen)
25:     return procCostTable
```

signal processing techniques from trend analysis and clustering approaches. The trend analysis generates a trend signal from the measured power waveform using a rolling median filter. A simple concept that the synchronization pattern has twice longer sleep (low-power) duration at start and end of the pattern than awake (high-power) in the middle, a median filter with a size slightly higher than three times of the awake duration will pull down the synchronization pattern into low-power values while pulling-up the measurements corresponds to the active NILM process into higher power as the NILM process has longer awake duration. Then, the probability distribution function (PDF) of the trend values giving lower peaks corresponds to low-power synchronization patterns and more than one higher peaks for the NILM active regions. Finally, active regions are extracted via clustering the measurement points into lower and higher power regions by applying a simple threshold technique. The first local minimum in the PDF gives the optimal decision threshold to cluster the regions. In severe signal corruption, some rarely misclassified points are adjusted in post-processing as they form tiny cluster sizes compared to adjacent clusters in the time series.

After effectively clustering the power data which corresponds to the NILM process and the background signals, the extraction proceeds into multilevel recursive synchronization pattern retrieval and removal from the WUM power data starting from the earliest level in the time-line, entire cost monitoring experiment, to the final level, multiprocessing scenarios, using the duration analysis of the consecutive synchronization patterns (see Algorithm 3).

The multilevel segregation process begins with loading settings (see line 3 in Algorithm 3). At the start, the system background steady-state power (idle state) is removed to extract the dynamic power components which correspond to active running processes (see line 4). Then, the decision threshold for data clustering is determined using a trend analysis (see lines 5 and 6) followed by clustering of the synchronization patterns and active windows (see line 7). It pro-

ceeds to extract the experiment repetition scenarios using total marker length of adjacent synchronizations ($syncDuration$) equals to $15 \times syncLen$ (see line 9), after the isolation of the granular region of interest, i.e., retrieve the entire cost monitoring section from the rest of the dynamic power data using $syncDuration >= 20 \times syncLen$ (see line 8). For each experiment repetition regions (see line 10 - 16), the different number of houses selection scenarios are extracted using $syncDuration = 10 \times syncLen$ marker length (see line 12) followed by retrieving the number of processes cases using $syncDuration = 5 \times syncLen$ (see line 15) for each previously extracted number of houses scenarios (see line 14). Eventually, the NILM ROIs data-set (see line 17) is prepared after labeling the regions with the number of experiment repetition, number of houses, and number of processes.

Algorithm 3 WUM Reading ROI Extraction

1: **procedure** ROIEXTRACTION(P_{WUM})
2: $P_{NILM} \leftarrow []$
3: $d \leftarrow LoadExpSetting()$ ▷ d is $syncLen$
4: $P_D, P_{BG} \leftarrow RemoveBgPower(P_{WUM})$
5: $trendValues \leftarrow TrendAnalysis(P_D, d)$
6: $th \leftarrow DetermineThr(trendValues)$ ▷ th is decision threshold
7: $clusterIds \leftarrow ClusteringTrend(trendValues, th)$
▷ NILM section extraction
8: $P^*_{NILM} \leftarrow GetROI(P_D, clusterIds, 20d)$
▷ Repeated experiment sections extraction
9: $P^*_{NILMExps}, expIds \leftarrow GetROI(P^*_{NILM}, clusterIds, 15d)$
10: **for** $expId$ *in* $expIds$: **do**
11: $P^*_{Exp} = SelectPow(P^*_{NILMExps}, expId)$
▷ Number houses of selection sections extraction
12: $P^*_{NILMNhs}, NhIds \leftarrow GetROI(P^*_{Exp}, clusterIds, 10d)$
13: **for** $NhId$ *in* $NhIds$: **do**
14: $P^*_{Nh} = SelectPow(P^*_{NILMNhs}, NhId)$
▷ Number of processors selection sections extraction
15: $P_{NILMNps}, NpIds \leftarrow GetROI(P^*_{Nh}, clusterIds, 5d)$
16: $P_{NILM} \leftarrow join(P_{NILM}, P_{NILMNps}, expId, NhId, NpIds)$
17: **return** P_{NILM}

$GetROI$ - extract region or section of interest
* - has data points belongs to synchronization at inner levels

E. Computation Cost Analysis

Once the active ROIs are extracted from the WUM log, processing time can be simply calculated from the duration of the windows. The WUM has used a 1 second sampling time. Thus, the PT can approximate by the size or length of power readings samples. To estimate the energy cost, we use a discrete-based method such as *trapezoidal* (Equation 1) to approximate the integral function of the power from the discrete power readings.

$$E = \int_a^b P(t)dt \approx \frac{h}{2}\left[P_0 + 2\sum_{i=1}^{n-1} P_i + P_n\right] \quad (1)$$

where E is the estimated energy cost during the time t window of $[a, b]$ while P is the instantaneous power. The P_i is i^{th} sample of the power reading, h is the sampling interval, i.e. 1 second, and n is the number of samples.

F. Predictive Modeling for Computation Cost

The cost prediction models are vital for providing relevant cost information in deploying a load disaggregation system. Such models are crucial in determining cost-efficient configuration and meet scalability requirements.

In this study, the cost data-set was split into training- and test-set when developing the cost models. Separate prediction models were trained for processing time and energy costs for a given number of houses and multiprocessing configuration. The models were validated with an out-of-bag set after training, which comprises a N_hs with larger load scales. To make the prediction models robust for in- and out-of-sample predictions, the targets were converted into per-unit house (puh) values by dividing them with the N_h. Interestingly, the puh values are converged into very closely overlapped curves which varies mainly with number of processes (see Figure 12 and 13 in Section V-A). Once the puh prediction is made using the models, the final cost estimation is calculated by multiply the result with the given N_h.

In our study, machine learning is employed to build the regression prediction models which capture the underlying relationships between the continuous value target cost variable and predictor features. Random Forest (RF) is one of the most popular and versatile machine learning algorithm based on bagged ensemble decision trees [46], [47]. Moreover, the performance of the RF models is compared with simple statistical-based models, Mean Estimation (ME) for each unique number of processors.

a) Statistical model: mean value of the puh costs for each N_p are calculated from the training set to be later used for prediction. It uses a simple assumption that the puh costs solely depend on the given N_p.

b) Machine learning model: a RF was trained to predict the puh costs using a predictor feature from N_p and normalized N_h. We employed a reciprocating technique, $1/N_h$, to normalize the number of houses to make sure the feature values bounded into the range of (0, 1]. Since the target cost functions are found to be not complex with a couple of features and small data-set, when selecting optimal hyperparameters configurations for the RF models, $Grid - search$ has been employed with a four-fold grouped cross-validation based on the load scales, N_h. We have used Scikit-Learn's Random Forests[3] to build the RF models using MSE (Mean Square Error) as a cost function. Some of the main hyperparameters used in the tuning are $no\_estimators$, max_depth and $min\_samples\_leaf$.

Appendix C provides more details on notations to elucidate the proposed methods with mathematical modeling.

V. RESULTS AND DISCUSSION

The term Cloud-based Online-NILM Algorithm (CbON) usually refers to the Online-NILM algorithm that runs on remote servers. Energy companies currently deploy data analytics applications such as CbON algorithms in their data centers either on dedicated servers or virtual machines and slowly shifting towards Cloud systems. Thus, to demonstrate the robustness of the proposed cost monitoring and modeling methodology, we run our experiment on two different use-cases, i.e., a Dedicated Server (DS) and a Cloud Virtual Server (CVS). The system specification of the computing systems is provided in Table IV. The purpose of these separate use-cases

[3]https://scikit-learn.org/stable/supervised_learning.html

is not to make a comparison between the two server systems but rather to demonstrate the robustness of the proposed cost monitoring and modeling methodology for NILM algorithm deployments on dedicated servers and Cloud virtual machines.

For prediction model validation at large scale scenarios, the load scale was varied from 4000 to 10000 houses while the remaining lower scales from 1000 to 2500 were utilized for model training. The experiment repetitions are ten and five times for the DS for the training- and test-set, respectively, while five and three times are used for the CVS due the limited storage capacity in the WUM and relatively lower computational power. Furthermore, as explained in Section IV-C, to minimize the interference in WUM measurement from other processes, the network connection of use-case of DS was disabled, while other virtual machines on the system were disabled for the use-case of CVS.

System	System Specification
DS	Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz, 4 Core(s), 8 Logical Processor(s) and 16GB RAM using on Windows 10 Pro Version 1803
CVS	Intel(R) Xeon(R) CPU E5-2697 v2 @ 2.70GHz, 2693 Mhz, 8 Core(s), 8 Logical Processor(s) and 32GB RAM using Windows 10 Pro Version 1903, running on Oracle VirtualBox 1.2 installed on a GenuineIntel GNU/Linux Server with 48 CPUs

TABLE IV: Systems used during experimentation

This section is composed of three major parts discussed in the subsequent subsections, i.e., extraction of computational power consumption of the NILM algorithm for different scenarios from WUM reading (ROI extraction), computation costs analysis, and predictive modeling of the computation costs. We concisely discuss results for both use-case servers, i.e., DS and CVS simultaneously.

A. Region of Interest Extraction from WUM Record Data

Using Algorithm 3, explained in Section IV-D, the results of the WUM power record data segregation, to extracted activity regions belonging to the NILM algorithm, are portrayed in Figure 4 - 10 for the two use-case servers. Initially, the dynamic power (corresponds to the active processes) is extracted by removing the background or static power consumption of the computing system (see Figure 4 for DS). Then, each regions which corresponds to each experiment scenarios are extracted using the extraction blocks incorporated in WUM record data segregation algorithm discussed in Section IV-D. In Figure 5 and 9, sample of these extracted Region of Interest (ROI) are depicted. Moreover, the process of isolation of the synchronization patterns and effectively determining the cluster decision threshold from the WUM power reading data using the trend analysis and cluster-based techniques are illustrated in Figure 6 and Figure 10. The WUM record data segregation algorithm robustly extracted the experimental sections in both the use-case servers through locating and removal of the synchronization signals.

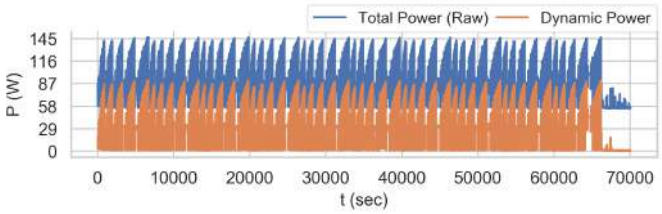
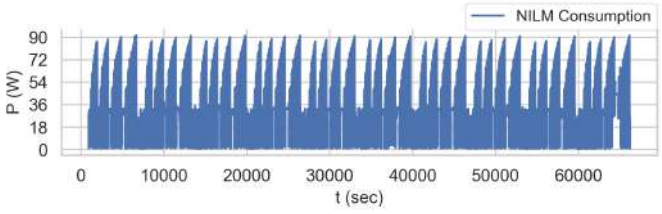
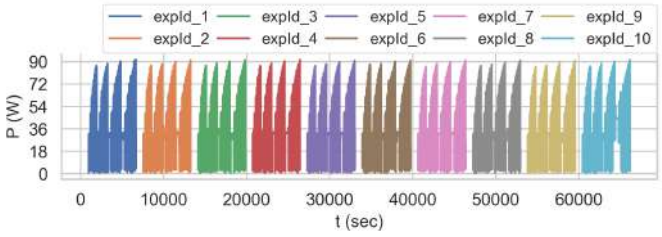


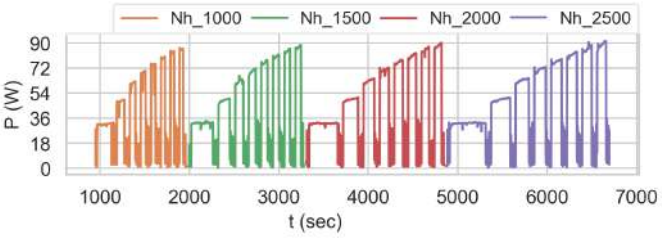
Fig. 4: Background power removal from WUM reading for DS



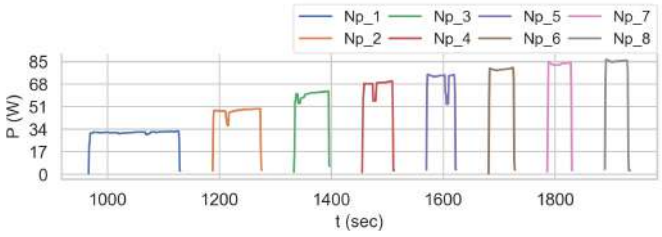
(a)



(b)



(c)



(d)

Fig. 5: ROI extraction for DS (a) NILM experiment extraction, (b) repeated experiment, (c) number houses selection on $expId = 1$, (d) and number of parallel processes on $N_h = 1000$

B. Computation Cost Analysis

The average computational processing time and energy costs for a given number of processors in multiprocessing and number of house variations in load scales are depicted in Figure 12 and 13 for the DS and CVS, respectively. The results demonstrate the costs are affected by the number of parallel processes and the number of house selections. To provide an

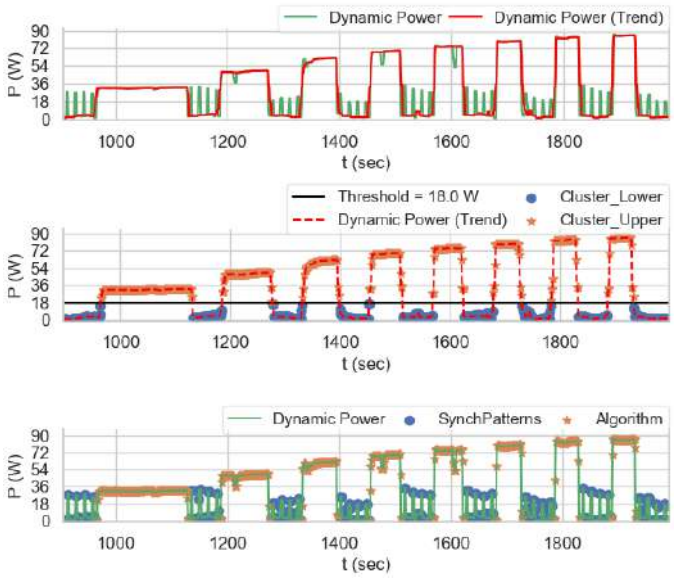


Fig. 6: WUM record data segregation at $N_h = 1000$ using trend analysis and clustering for DS

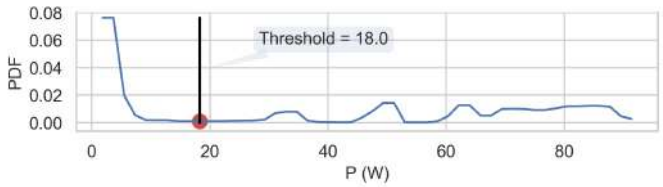


Fig. 7: Trend PDF and clustering threshold for DS

intuitive analysis, per-unit house (puh) normalized costs are portrayed in (c) and (d) subplots, respectively. The per-unit house cost follows exponential decaying patterns and almost utterly depends on the N_p selections. In the CVS, the costs vary across the different number of processors in a slight discrete manner. The impact of N_p selection is more pronounced on the computational energy, which can be quantified by the higher *Pearson Correlation* -0.46 as compare to -0.15 for the DS. This is might be owing to the architectural difference: CVS has eight logical processors from eight cores; while the DS has eight processors from four cores (see Table IV).

Moreover, an interesting comparison of PT from WUM and PT_{code} measured using the within a code Python time library. They are consistently equivalent with less than 0.5% difference in the average across all the multiprocessing scenarios (see Figure 14 and Figure 15). This validates the usage of system-level based measurement with robust ROI extraction techniques for an algorithm computational cost analysis.

In the DS use-case, the computational costs are significantly reduced using multiprocessing up-to by 75% for PT. At the same time a lower gain (because power consumption increases with additional processors) was achieved around 33% for the energy consumption as compared with a single processor.

In use-case of CVS, the costs are immensely curtailed using multiprocessing up-to 86% for PT while 68% leverage was achieved for energy cost compared with a single processor. The CVS has longer PT which causes a higher PE than DS due to the slower processing capability of its CPUs.

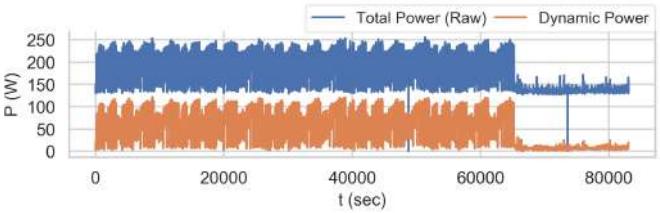
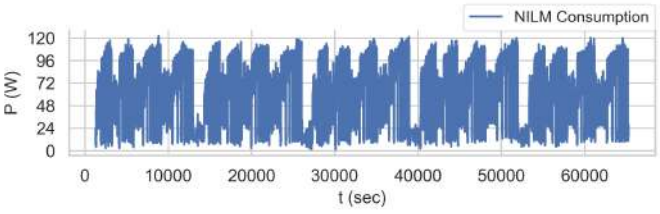
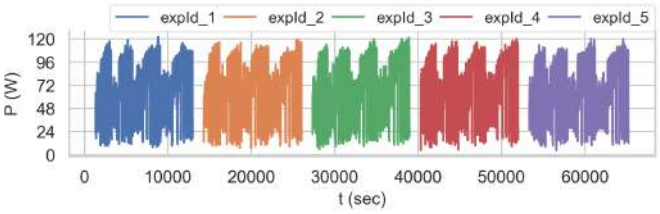


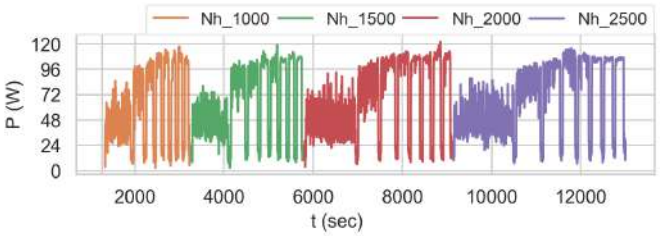
Fig. 8: Background power removal from WUM reading for CVS



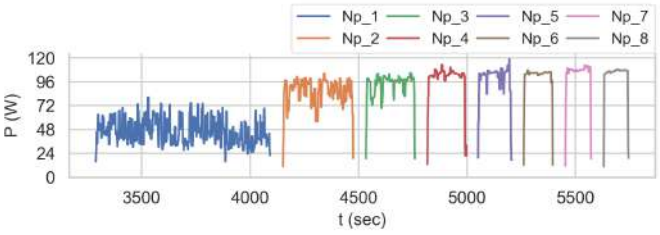
(a)



(b)



(c)



(d)

Fig. 9: ROI extraction for CVS (a) NILM experiment extraction, (b) repeated experiment, (c) number houses selection on $expId = 1$, (d) and number of parallel processes on $N_h = 1500$

When the energy cost is analyzed relative to the median one-hour disaggregated house energy load (255 watt-hour) by the NILM algorithm, the relative cost is below 0.0006% for the DS and 0.0028% for the CVS use-case (see Table V). When the idle or static energy consumption of the DS system, 54 watt-hour, is also considered, the energy cost increases to around 0.0015% for the maximum number of houses ($N_{h\_capacity} = 22,216$) that can be processed during the one-

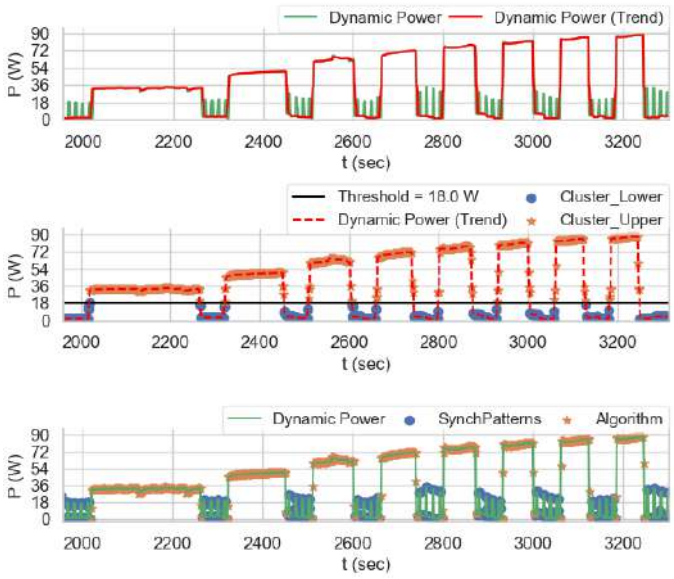


Fig. 10: WUM record data segregation at $N_h = 1500$ using trend analysis and clustering for CVS

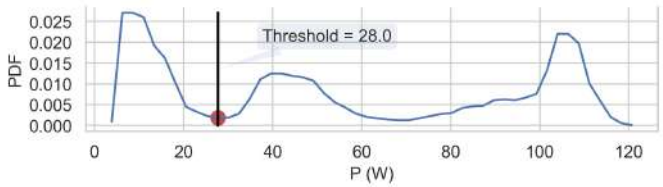


Fig. 11: Trend PDF and clustering threshold for CVS

hour dw with a single processor ($N_p = 1$). The load scale capacity is estimated from the PT_{puh} prediction model for a given N_p and dw (see Equation in Table V). The cost can be reduced to 0.0006% while significantly enhancing the load capacity to $89,479$ houses if multiprocessing with $N_p = 8$ are utilized. The CVS has predicted load scale capacity of 6,523 and 46,210 along with energy cost of 0.0104% and 0.0020% for $N_p = 1$ and $N_p = 8$ respectively. This is mainly due to the higher static power, 127 watts, and slower processing.

Finally, though the actual energy consumption depends on the underlying system static and dynamic powers, CPU processing speed, and multiprocessing implementation, the experimental results demonstrate that the energy cost of the NILM algorithms are diminutive as compared to the expected 12% [3], [4] energy efficiency saving in households from using NILM systems. However, energy-saving from load monitoring tools creates awareness of energy usage via a shift to efficient appliances and user behavior changes. This means the actual efficiency gain from the tools exacerbates once the user makes the appropriate actions although it may be in the long run. On the other hand, the computation energy cost of the monitoring system become relatively significant due to lower efficiency gain for the users. The effect on carbon emission from the computing system also becomes more pronounced for large load scales deployment. Thus, mindful consideration of multiprocessing, especially with multiple physical cores than logical ones, needs to be given to leverage performance in significantly enhancing the load capacity of the computing system

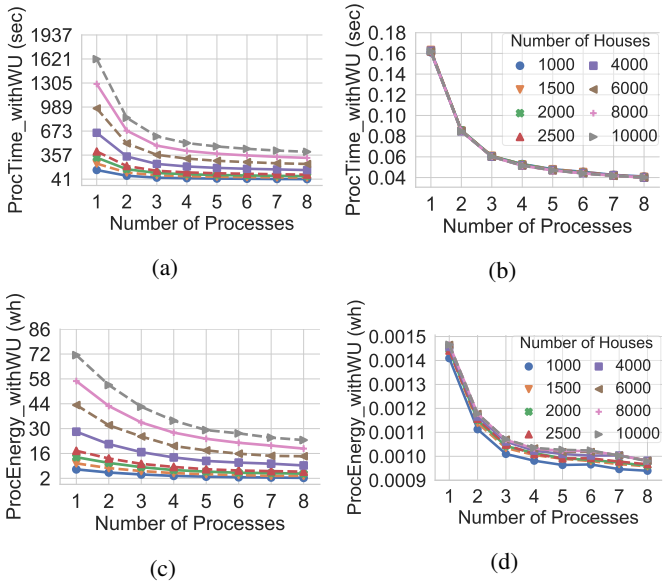


Fig. 12: Computational costs for DS: PT (a), PT_{puh} (b), PE (c) and PE_{puh} (d)

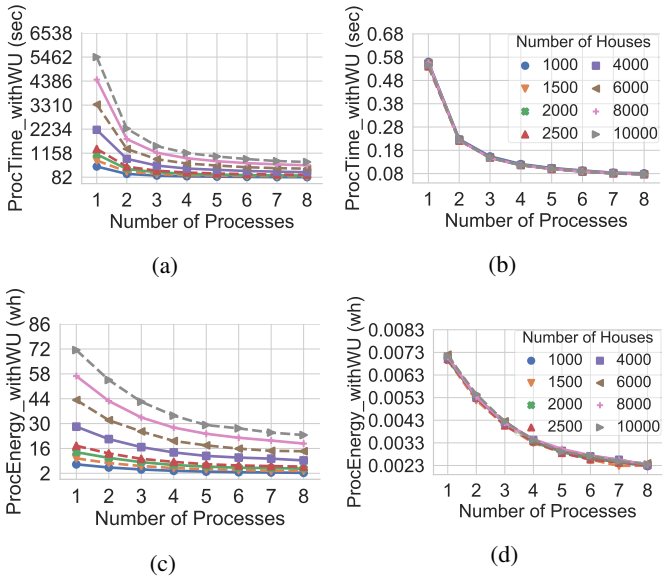


Fig. 13: Computational costs for CVS: PT (a), PT_{puh} (b), PE (c) and PE_{puh} (d)

while optimizing the computational energy consumption.

C. Predictive Modeling for Computation Cost

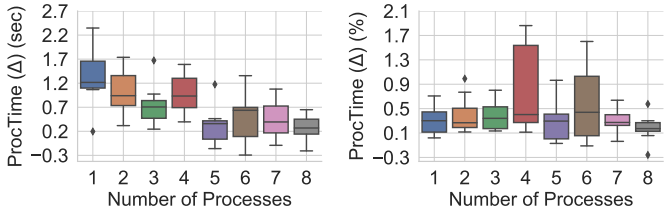
In this section, the prediction performance of the cost models from statistical and machine learning models on the test-set is discussed. The performance metrics employed to evaluate the prediction cost models are provided in Table VI.

The prediction evaluation demonstrates that the computational costs of the NILM algorithm can be modeled effectively with very high accuracy (see Table VII and VIII, and Appendix provide details). This validates the promising applicability of the data-driven modeling approach without detailed knowledge of the underlying running hardware and operating system. Generally, the machine learning based regression model using

	NILM PE_{puh}											
Average House Load in dw (wh)	**DS (wh)**			**CVS (wh)**			**DS (%)**			**CVS (%)**		
	$N_p = 1$	$N_p = 4$	$N_p = 8$	$N_p = 1$	$N_p = 4$	$N_p = 8$	$N_p = 1$	$N_p = 4$	$N_p = 8$	$N_p = 1$	$N_p = 4$	$N_p = 8$
255	0.0014	0.0010	0.0010	0.0071	0.0034	0.0023	0.0006%	0.0004%	0.0004%	0.0028%	0.0013%	0.0009%
$N_{h\_capacity} = \frac{DW}{PT_{puh}(N_p)}$	**NILM PT_{puh}**						**Maximum N_h can be processed within dw = 1 hour**					
	DS (sec)			**CVS (sec)**			**DS ($N_{h\_capacity}$)**			**CVS ($N_{h\_capacity}$)**		
	$N_p = 1$	$N_p = 4$	$N_p = 8$	$N_p = 1$	$N_p = 4$	$N_p = 8$	$N_p = 1$	$N_p = 4$	$N_p = 8$	$N_p = 1$	$N_p = 4$	$N_p = 8$
	0.1620	0.0521	0.0402	0.5519	0.1170	0.0779	22216	69077	89479	6523	30763	46210
	NILM for $N_{h\_capacity}$+ Static Energy within dw = 1 hour											
Average House Load in dw (wh)	**DS (wh)**			**CVS (wh)**			**DS (%)**			**CVS (%)**		
	$N_p = 1$	$N_p = 4$	$N_p = 8$	$N_p = 1$	$N_p = 4$	$N_p = 8$	$N_p = 1$	$N_p = 4$	$N_p = 8$	$N_p = 1$	$N_p = 4$	$N_p = 8$
$255 \times N_{h\_capacity}$	86.190	124.251	140.705	173.043	231.256	234.439	0.0015%	0.0007%	0.0006%	0.0104%	0.0029%	0.0020%

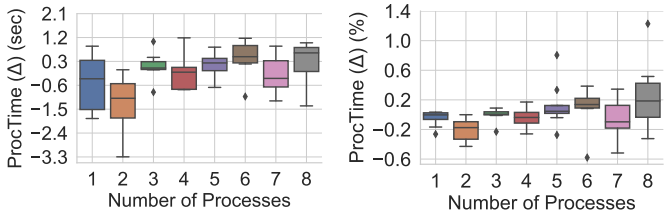
* Static energy cost for the DS is 54 wh whereas CVS consumes 127 wh.

TABLE V: NILM computational cost analysis as compared to average house energy load



(a) (b)

Fig. 14: Comparison of PT for DS measured from WUM and code-based, difference (a) and in percentage (b)



(a) (b)

Fig. 15: Comparison of PT for CVS measured from WUM and code-based, difference (a) and in percentage (b)

RF, leveraged with the additional feature from N_h, outperforms the average statistical mode. Generally, the PE models is under-perform than the PT models. This is because the energy is a product of two variables, i.e., power and time, and its value is influenced by the measurement uncertainty in the processing time as well as in the power consumption. The overall prediction errors are higher for the CVS. This was expected as the measurement of the CVS is noisier than the DS, as well as a lower number of experiment repetition, was used for CVS due to longer processing time and storage capacity limitation of WUM as discussed in the introductory paragraph of Section V-A.

D. Impact of Sampling Rate on Computational Cost

We have briefly discussed below the inferred insights of the costs of NILM algorithms based on the sampling rate and disaggregation time-window selections.

A higher load sampling rate tends to achieve better disaggregation performance as more load signatures, such as transient events and electrical noise, can be retrieved at high frequencies. Higher sampling frequencies in the range of $10 - 100MHz$ are usually required as compared to $\leqslant 1Hz$ for low sampling rates, to achieve the promised accuracy. At these high ranges, the cost of data size increases along with the requirement for expensive power meters due to sophisticated hardware [2]. For CbON, the size of the reading segment may have a significant impact on data transmission as the transfer

Metric	**Equation**
Error	$e_i = y_i - t_i$
Absolute Percentage Error (APE)	$ep_i = \left\lvert \frac{e_i}{y_i} \right\rvert$
Root Mean Square Error	$RMSE = \sqrt{\frac{1}{N}\sum_{i=0}^{N} e_i^2}$
Mean APE	MAPE= $\frac{1}{N}\sum_{i=1}^{N} ep_i$
Maximum APE	$MaxAPE = Max\{\forall ep_i\}$
Confidence Interval of APE (CI)	$CI_\alpha = P(ep \leqslant \alpha)$
Regression Fit Score ($R-value$)	$R = \frac{\sum_{i=1}^{N}(y_i-\bar{y})(t_i-\bar{t}))}{\sum_{i=1}^{N}(y_i-\bar{y})^2 \sum_{i=1}^{N}(t_i-\bar{t})^2}$

*Where y_i and i_t are the actual and estimated values of the i^{th} sample while , N is the sample size, P is Probability Density Function of ep, and α is a threshold of the error confidence interval

TABLE VI: Performance metrics

Processing Time				
Metric	P_{ME}	P_{RF}	$Gain$	$Gain_{\%}$
$RMSE$ (sec)	5.02	3.17	-1.85	-36.80
$MAPE$ (%)	1.12	0.63	-0.49	-43.93
$MaxAPE$ (%)	2.61	1.93	-0.67	-25.82
$CI_{1\%}$	53.13	78.13	25.00	47.06
$CI_{3\%}$	100.00	100.00	0.00	0.00
$CI_{5\%}$	100.00	100.00	0.00	0.00
$R - value$	1.00	1.00	0.00	0.00
Processing Energy				
$RMSE$ (wh)	0.26	0.18	-0.07	-28.08
$MAPE$ (%)	3.03	2.09	-0.94	-31.04
$MaxAPE$ (%)	4.26	3.50	-0.76	-17.76
$CI_{3\%}$	46.88	90.63	43.75	93.33
$CI_{5\%}$	100.00	100.00	0.00	0.00
$R - value$	1.00	1.00	0.00	0.00

* $Gain = P_{RF} - P_{ME}$ where P_{ME} and P_{RF} are performance score of ME and RF respectively, and $Gain_{\%} = (Gain \times 100)/P_{ME}$.

TABLE VII: Cost prediction performance for DS

Processing Time				
Metric	P_{ME}	P_{RF}	$Gain$	$Gain_{\%}$
$RMSE$ (sec)	18.38	16.89	-1.50	-8.14
$MAPE$ (%)	1.28	1.08	-0.20	-15.75
$MaxAPE$ (%)	3.23	2.83	-0.40	-12.44
$CI_{1\%}$	43.75	53.13	9.38	21.43
$CI_{3\%}$	96.88	100.00	3.13	3.23
$CI_{5\%}$	100.00	100.00	0.00	0.00
$R-value$	1.00	1.00	0.00	0.00
Processing Energy				
$RMSE$ (wh)	0.85	0.82	-0.03	-3.63
$MAPE$ (%)	2.81	2.54	-0.26	-9.30
$MaxAPE$ (%)	6.35	5.35	-1.00	-15.74
$CI_{1\%}$	15.63	18.75	3.13	20.00
$CI_{3\%}$	62.50	68.75	6.25	10.00
$CI_{5\%}$	84.38	93.75	9.38	11.11
$CI_{10\%}$	100.00	100.00	0.00	0.00
$R-value$	1.00	1.00	0.00	0.00

* $Gain = P_{RF} - P_{ME}$ where P_{ME} and P_{RF} are performance score of ME and RF respectively, and $Gain_{\%} = (Gain \times 100)/P_{ME}$.

TABLE VIII: Cost prediction performance for CVS

needs to be done frequently for real-time load disaggregation on remote servers. Moreover, the number of data points in the given disaggregation window has a proportional impact on the computation of FHMM based disaggregation algorithms.

Furthermore, the choice of disaggregation time-window also affects the computational cost and disaggregation accuracy. An hour window was chosen in our experiments to provide a good quasi-real-time disaggregation. The one-hour window produces good accuracy while keeping the cost lower because of the low probability for user activity changes in smaller time-windows. Smaller time-windows may have a higher running cost as the cost due to an increase in the entire disaggregation algorithm calls outweighs the complexity reduction due to the relatively shorter sequence length in the FHMM. Moreover, as appliance features such as duration in HMM-based algorithms become less distinctive, the guarantee for disaggregation accuracy leverage is limited for smaller windows [21]. A sliding-window function is often invoked to update results when an additional reading segment arrives for accuracy improvement, which introduces an extra computational cost.

VI. Limitations

This section briefly describes the limitations and the possible cause of errors that could affect our work.

The computational expenses of the Cloud-based Online-NILM Algorithm are not confined only to the execution of the algorithm. The cost associated with the transmission of the reading data from the smart-meter into the Cloud data store is also non-trivial. In our analyses, we found ingestion of the disaggregation results back to the SQL database takes a significant portion of the analyzed execution time. However, our aim is instead to provide granular estimations for a NILM algorithm.

Furthermore, system-level power measurement is susceptible to interference noise of other active processes running in the system besides the target algorithm. Hence, the computing system needs to be stabilized to minimize the noise from corrupting the WUM measurement. Security applications such as Window Defender and Anti-virus software, as well as an active network connection, were observed to introduce a noise during our experimentation. Disabling other active processes and repeating the experiment a few times, and then averaging the measurements can be feasible mitigation techniques.

Owing to the sampling rate constraint of the power measuring devices such as WUM, the processing time of the target algorithm execution requires to be long enough, spanning multiple samples. Moreover, the execution duration needs to be longer than the filter size of the trend analysis to effectively spot synchronization patterns. In our case, the longer processing duration is achieved by disaggregated loads from multiple houses in a single run.

We have normalized the impact of the number of houses into per-unit values, and the cost prediction models are effective for the out-of-bag unseen larger number of houses in the test cases. However, normalization of the number of processors is more challenging, and the behavior is particular to the computing systems. Defining the maximum parallel processing from the system's number of logical processors is recommended, and including this range into the training set voids the normalization difficulty for dedicated servers except during upgrading. Nevertheless, the number of allocated processors can be extended easily in Cloud Virtual Machines, and retraining of the models for the unseen number of processors is essential.

VII. Conclusion and Recommendations

In this study, we present generic data-driven cost monitoring, analyzing, and modeling approaches from system-level instant power consumption. The proposed methods were tested and validated on Dedicated Server and Cloud Virtual Server systems executing a Cloud-based Online-NILM Algorithm for real-time load disaggregation.

The study demonstrates exciting and accurate results in analyzing and modeling the computational time and energy expenses associated with a NILM algorithm at large load scales from thousands of households. The study will play a crucial role in providing relevant decision insights that ameliorate effective planning and deployment strategies of load monitoring tools considering the computational energy and processing time costs in large scale scenarios. Furthermore, our study will be vital for encouraging the development of load monitoring algorithms to consider computation cost efficiency perspective besides their monitoring efficacy.

References

[1] G. W. Hart, "Nonintrusive appliance load monitoring," *Proceedings of the IEEE*, vol. 80, no. 12, pp. 1870–1891, 1992.

[2] A. Zoha, A. Gluhak, M. Imran, and S. Rajasegarar, "Non-intrusive load monitoring approaches for disaggregated energy sensing: A survey," *Sensors*, vol. 12, no. 12, pp. 16 838–16 866, 2012.

[3] C. Klemenjak and P. Goldsborough, "Non-intrusive load monitoring: A review and outlook," *arXiv preprint arXiv:1610.01191*, 2016.

[4] K. C. Armel, A. Gupta, G. Shrimali, and A. Albert, "Is disaggregation the holy grail of energy efficiency? the case of electricity," *Energy Policy*, vol. 52, pp. 213–234, 2013.

[5] H. Acar, G. Alptekin, J.-P. Gelas, and P. Ghodous, "The impact of source code in software on power consumption," 2016.

[6] G. Da Costa, J.-M. Pierson, and L. Fontoura-Cupertino, "Mastering system and power measures for servers in datacenter," *Sustainable Computing: Informatics and Systems*, vol. 15, pp. 28–38, 2017.

[7] A. Noureddine, A. Bourdon, R. Rouvoy, and L. Seinturier, "A preliminary study of the impact of software engineering on greenit," in *2012 First International Workshop on Green and Sustainable Software (GREENS)*. IEEE, 2012, pp. 21–27.

[8] R. Bertran, M. Gonzalez, X. Martorell, N. Navarro, and E. Ayguade, "Decomposable and responsive power models for multicore processors using performance counters," in *Proceedings of the 24th ACM International Conference on Supercomputing*, 2010, pp. 147–158.

[9] S. Wang, H. Chen, and W. Shi, "Span: A software power analyzer for multicore computer systems," *Sustainable Computing: Informatics and Systems*, vol. 1, no. 1, pp. 23–34, 2011.

[10] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya, "Virtual machine power metering and provisioning," in *Proceedings of the 1st ACM symposium on Cloud computing*, 2010, pp. 39–50.

[11] S. Collange, D. Defour, and A. Tisserand, "Power consumption of gpus from a software perspective," in *International Conference on Computational Science*. Springer, 2009, pp. 914–923.

[12] T. Vogelsang, "Understanding the energy consumption of dynamic random access memories," in *43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2010, pp. 363–374.

[13] I. M. Murwantara and B. Bordbar, "A simplified method of measurement of energy consumption in cloud and virtualized environment," in *Fourth International Conference on Big Data and Cloud Computing*. IEEE, 2014, pp. 654–661.

[14] I. M. Murwantara and P. Yugopuspito, "Evaluating energy consumption in a different virtualization within a cloud system," in *2018 4th International Conference on Science and Technology (ICST)*. IEEE, 2018, pp. 1–6.

[15] P. Bartalos and M. B. Blake, "Green web services: Modeling and estimating power consumption of web services," in *19th International Conference on Web Services*. IEEE, 2012, pp. 178–185.

[16] D. Molaro, H. Payer, and D. Le Moal, "Tempo: Disk drive power consumption characterization and modeling," in *13th International Symposium on Consumer Electronics*. IEEE, 2009, pp. 246–250.

[17] L. Ardito, R. Coppola, M. Morisio, and M. Torchiano, "Methodological guidelines for measuring energy consumption of software applications," *Scientific Programming*, vol. 2019, no. 1, pp. 1–16, 2019.

[18] A. Noureddine, R. Rouvoy, and L. Seinturier, "A review of energy measurement approaches," *SIGOPS Oper. Syst. Rev.*, vol. 47, no. 3, pp. 42–49, nov 2013.

[19] J. M. Hirst, J. R. Miller, B. A. Kaplan, and D. D. Reed, "Watts up? pro ac power meter for automated energy recording," 2013.

[20] S. Atiewi, A. Abuhussein, and M. A. Saleh, "Impact of virtualization on cloud computing energy consumption: Empirical study," in *Proceedings of the 2nd International Symposium on Computer Science and Intelligent Control*, 2018, pp. 1–7.

[21] M. A. Mengistu, A. A. Girmay, C. Camarda, A. Acquaviva, and E. Patti, "A cloud-based on-line disaggregation algorithm for home appliance loads," *IEEE Transactions on Smart Grid*, vol. 10, no. 3, pp. 3430–3439, 2018.

[22] Q. Chen, P. Grosso, K. van der Veldt, C. de Laat, R. Hofman, and H. Bal, "Profiling energy consumption of vms for green cloud computing," in *Ninth International Conference on Dependable, Autonomic and Secure Computing*. IEEE, 2011, pp. 768–775.

[23] A. Hindle, A. Wilson, K. Rasmussen, E. J. Barlow, J. C. Campbell, and S. Romansky, "Greenminer: A hardware based mining software repositories software energy consumption framework," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. ACM, 2014, pp. 12–21.

[24] R. A. Giri, I. I. Staff Engineer, and A. Vanchi, "Increasing data center efficiency with server power measurements," *Document. Intel Information Technology. IT@ Intel White Paper*, 2010.

[25] M. Dong and L. Zhong, "Self-constructive high-rate system energy modeling for battery-powered mobile systems," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '11, 2011, p. 335–348.

[26] R. Mittal, A. Kansal, and R. Chandra, "Empowering developers to estimate app energy consumption," in *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking*, ser. Mobicom '12, 2012, p. 317–328.

[27] A. A. Nacci, F. Trovò, F. Maggi, M. Ferroni, A. Cazzola, D. Sciuto, and M. D. Santambrogio, "Adaptive and flexible smartphone power modeling," *Mobile Networks and Applications*, vol. 18, no. 5, pp. 600–609, Oct 2013.

[28] D. Di Nucci, F. Palomba, A. Prota, A. Panichella, A. Zaidman, and A. De Lucia, "Software-based energy profiling of android apps: Simple, efficient and reliable?" in *International conference on software analysis, evolution and reengineering (SANER)*. IEEE, 2017, pp. 103–114.

[29] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang, "Fine-grained power modeling for smartphones using system call tracing," in *Proceedings of the Sixth Conference on Computer Systems*, ser. EuroSys '11. ACM, 2011, pp. 153–168.

[30] T. Do, S. Rawshdeh, and W. Shi, "ptop: A process-level power profiling tool," 2009.

[31] M. Berges, E. Goldman, H. S. Matthews, and L. Soibelman, "Training load monitoring algorithms on highly sub-metered home electricity consumption data," *Tsinghua Science and Technology*, vol. 13, no. S1, pp. 406–411, 2008.

[32] Y. Wang, X. Wang, M. Chen, and X. Zhu, "Partic: Power-aware response time control for virtualized web servers," *IEEE Transactions on parallel and distributed systems*, vol. 22, no. 2, pp. 323–336, 2010.

[33] J. Kelly and W. Knottenbelt, "The uk-dale dataset, domestic appliance-level electricity demand and whole-house demand from five uk homes," *Scientific data*, vol. 2, p. 150007, 2015.

[34] P. Shenavar and E. Farjah, "Novel embedded real-time nilm for electric loads disaggregating and diagnostic," in *EUROCON 2007-The International Conference on Computer as a Tool*. IEEE, 2007, pp. 1555–1560.

[35] M. Berges, E. Goldman, H. S. Matthews, L. Soibelman, and K. Anderson, "User-centered nonintrusive electricity load monitoring for residential buildings," *Journal of computing in civil engineering*, vol. 25, no. 6, pp. 471–480, 2011.

[36] S. Makonin, F. Popowich, I. V. Bajić, B. Gill, and L. Bartram, "Exploiting hmm sparsity to perform online real-time nonintrusive load monitoring," *IEEE Transactions on Smart Grid*, vol. 7, no. 6, pp. 2575–2585, 2015.

[37] O. Krystalakos, C. Nalmpantis, and D. Vrakas, "Sliding window approach for online energy disaggregation using artificial neural networks," in *Proceedings of the 10th Hellenic Conference on Artificial Intelligence*. ACM, 2018, p. 7.

[38] H. Kim, M. Marwah, M. Arlitt, G. Lyon, and J. Han, "Unsupervised disaggregation of low frequency power measurements," in *Proceedings of the SIAM international conference on data mining*. SIAM, 2011, pp. 747–758.

[39] J. Z. Kolter and T. Jaakkola, "Approximate inference in additive factorial hmms with application to energy disaggregation," in *Artificial Intelligence and Statistics*, 2012, pp. 1472–1482.

[40] O. Parson, "Unsupervised training methods for non-intrusive appliance load monitoring from smart meter data," Ph.D. dissertation, University of Southampton, 2014.

[41] S. Drenker and A. Kader, "Nonintrusive monitoring of electric loads," *IEEE Computer Applications in Power*, vol. 12, no. 4, pp. 47–51, 1999.

[42] E. Patti, E. Pons, D. Martellacci, F. B. Castagnetti, A. Acquaviva, and E. Macii, "Multiflex: Flexible multi-utility, multi-service smart metering architecture for energy vectors with active prosumers," in *SMARTGREENS*. IEEE, 2015, pp. 1–6.

[43] J. Z. Kolter and M. J. Johnson, "Redd: A public data set for energy disaggregation research," in *SIGKDD*, vol. 25, 2011, pp. 59–62.

[44] G. Zaccone, *Python parallel programming cookbook*. Packt Publishing Ltd, 2015.

[45] S. Salvador and P. Chan, "Toward accurate dynamic time warping in linear time and space," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 561–580, 2007.

[46] A. Jaiantilal, Y. Jiang, and S. Mishra, "Modeling cpu energy consumption for energy efficient scheduling," in *Proceedings of the 1st Workshop on Green Computing*. ACM, 2010, pp. 10–15.

[47] J. Chen, B. Li, Y. Zhang, L. Peng, and J.-k. Peir, "Statistical gpu power analysis using tree-based methods," in *International Green Computing Conference and Workshops*. IEEE, 2011, pp. 1–6.

Mulugeta Weldezgina Asres is a Graduate Researcher at the Control and Computer Engineering Department of Politecnico di Torino. He accomplished his B.Sc. in Electrical and Computer Engineering and M.Sc. in Computer Engineering at EiT-M, Mekelle University. He conducted his Master Thesis and Post-graduate research on AI-powered NILM for complex systems at Midori Srl, Italy. His research interest revolves around Data-Driven AI-Models, Machine Learning, Deep Learning, Non-intrusive Load Monitoring, Internet of Things and Industry 4.0.

Luca Ardito is an Assistant Professor at the Department of Control and Computer Engineering at Politecnico di Torino, where he works in the Software Engineering research group. He received B.Sc., M.Sc., and Ph.D. in Computer Engineering from Politecnico di Torino. His current research interests are mobile development and testing, green software, new programming language analysis, and empirical software engineering methodologies. He is an associate editor of IEEE Access since November 2019, and he is the Young Professional representative for the IEEE Italy section since February 2020.

Edoardo Patti (M'16) is Assistant Professor at Politecnico di Torino. He received both M.Sc. and Ph.D. degrees in Computer Engineering at Politecnico di Torino in 2010 and 2014, respectively. His research interests concern: i) Ubiquitous Computing and Internet of Things; ii) Smart Systems, Cities and Mobility; iii) Software architectures with particular emphasis on infrastructure for Ambient Intelligence; iv) Software solutions for simulating and optimising energy systems; v) Software solutions for energy data visualisation to increase user awareness.