An Approach Using Performance Models for Supporting Energy Analysis of Software Systems

Vincenzo Stoico[1][0000−0002−3681−372X], Vittorio Cortellessa[1][0000−0002−4507−464X], Ivano Malavolta[2][0000−0001−5773−8346], Daniele Di Pompeo[1][0000−0003−2041−7375], Luigi Pomante[1][0000−0002−4137−3634], and Patricia Lago[2][0000−0002−2234−0845]

[1] Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Italy
vincenzo.stoico@graduate.univaq.it
{vittorio.cortellessa, daniele.dipompeo, luigi.pomante}@univaq.it
[2] Department of Computer Science, Vrije Universiteit Amsterdam, The Netherlands
{i.malavolta, p.lago}@vu.nl

Abstract. Measurement-based experiments are a common solution for assessing the energy consumption of complex software systems. Since energy consumption is a metric that is sensitive to several factors, data collection must be repeated to reduce variability. Moreover, additional rounds of measurements are required to evaluate the energy consumption of the system under different experimental conditions. Hence, accurate measurements are often unaffordable because they are time-consuming. In this study, we propose a model-based approach to simplify the energy profiling process and reduce the time spent performing it. The approach uses Layered Queuing Networks (LQN) to model the scenario under test and examine the system behavior when subject to different workloads. The model produces performance estimates that are used to derive energy consumption values in other scenarios. We have considered two systems while serving workloads of different sizes. We provided 2K, 4K, and 8K images to a Digital Camera system, and we supplied bursts of 75 to 500 customers for a Train Ticket Booking System. We parameterized the LQN with the data obtained from short experiment and estimated the performance and energy in the cases of heavier workloads. Thereafter, we compared the estimates with the measured data. We achieved, in both cases, good accuracy and saved measurement time. In case of the Train Ticket Booking System, we reduced measurement time from 5 hours to 35 minutes by exploiting our model, this reflected in a Mean Absolute Percentage Error of 9.24% in the estimates of CPU utilization and 8.72% in energy consumption predictions.

Keywords: Layered Queuing Networks · Performance Analysis · Energy Consumption · Software

1 Introduction

The ubiquity of ICT devices triggered a continuous digitalization of information, thus facilitating access, storage, and manipulation of data. ICT brought several benefits to society, such as monitoring the health conditions of people in real-time or having almost universal access to educational content. However, continuous digitization has also downsides. A considerable amount of information demands expensive resources for processing and storage, with a consequent rising need for energy to build and power ICT devices. As energy demand increases, the impact of ICT in terms of carbon dioxide (CO_2) emissions becomes significant [21]. Belkhir et al. estimate that ICT devices will produce 14% of global CO_2 emissions by 2040 [5].

As already discussed in 2018 by Georgiou et al [15] in the context of IoT systems, technology has made considerable advancements for increasing hardware power consumption savings, which however can be undermined by poor design decisions at the software level. Software energy optimization is a hard endeavor, where multiple (and frequently conflicting) design and implementation decisions can influence the energy footprint of the software [21]. Making developers aware of the impact of their decisions on the energy consumption of their software is fundamental to cutting it down [10]. However, energy optimization cannot be pursued in isolation, because it may negatively impact on other non-functional attributes of software and, in particular, on performance. Hence, software design decisions have to induce acceptable tradeoffs between the satisfaction of performance requirements and power consumption savings [12]. Energy/performance tradeoffs can be analyzed by measurement-based experiments, although they can be very time-consuming and they need contextual conditions to be taken under control (e.g., temperature of devices) for achieving reliable results. Modeling is often a valuable alternative to measurements, especially in cases where enough information about the software system and its context is known. Obviously, energy/performance models have to hold an appropriate level of abstraction that allows practitioners not to miss relevant aspects of the system and, at the same time, to keep an acceptable complexity of the model evaluation process [6].

This study explores the combination of measurement-based experiments and modeling in the context of energy/performance analysis of software systems, in order to benefit from the advantages of both of them. In particular, we investigate how to exploit performance models (specifically, Layered Queuing Networks – LQNs) to reduce the experimentation time while keeping a high accuracy in the energy consumption estimate of a software system. Although LQNs are a well-known modeling notation for software performance analysis [13, 20], their adoption for energy consumption analysis has yet to be developed. LQNs fit our purposes as they represent system resources as time-consuming entities. As energy consumption is a time-based metric, it is possible to define a relationship between performance and energy consumption according to resource utilization (namely, the amount of time a resource is busy). We reduce the reality gap between the LQN model and the system under analysis by systematically refining the LQN model using data obtained from a small-scale measurement-based ex-

periment. After achieving satisfactory accuracy, the LQN can be used to study the system under different workloads and get corresponding resource utilization estimates. These estimates can be multiplied by the energy consumed per second by each resource to obtain the total energy consumed while the resources are busy. We tested our approach on two different systems: Digital Camera, which we employ as a running example, and Train Ticket Booking System. The former is an image processing application that we deployed on an embedded platform, while the latter is a container-based web application for managing train bookings. For the Digital Camera, the supplied workloads correspond to batches of images of different resolutions, namely 2K, 4K, and 8K. Instead, for Train Ticket Booking System the workloads consists of bursts of 75, 150, 225, 300, 375, 450, and 500 customers. We parametrized the LQN with data measured for the batch of 2K images and the 75-customer burst, respectively, then we estimated resource utilization and energy consumption for different scenarios. Promising results emerged by comparing the measured data with the estimates. The Mean Absolute Percentage Error (MAPE) obtained for Train Ticket Booking System equals 9.24% for CPU Utilization and 8.47% for energy consumption. At the same time, we reduced experimentation time from 5 hours to 35 minutes.

Hence, the main contributions of this study are: (i) an approach for using LQNs to make accurate energy estimations of a software system, (ii) a preliminary empirical evaluation of the proposed approach on two different software systems across different domains, (iii) a replication package for the independent verification and replication of the performed evaluation [3]. The paper is structured as follows. Section 2 presents energy consumption basics and describes the Layered Queuing Networks. Section 3 delves into the approach and shows it through a running example: the Digital Camera case study. Section 4 outlines the experiments and the results achieved on the Train Ticket Booking System case study. Threats to validity and related work are discussed, respectively, in Section 5 and Section 6. The paper ends with conclusions in Section 7.

2 Background

2.1 Software Energy Measurement

The physical quantities used for expressing the energy consumed by software executions are electrical energy and electrical power. Electrical energy quantifies the amount of work needed to drive current through a circuit, while electrical power refers to the rate the energy is consumed by the circuit at any instant. Energy is commonly measured in joule (J), which is defined in the International System of Units (SI). Power, unlike energy, expresses a rate. Indeed, power is defined as the total energy consumed over time measured in joule per second ($\frac{J}{s}$) or, following the standard SI, in watt (W).

Literature includes a plethora of tools for measuring software energy consumption [9]. We distinguish between energy profilers and power monitors. Energy profilers are software tools providing an estimation of the energy consumed

[3] https://doi.org/10.5281/zenodo.7877782

by a running application. Compared with power monitors, energy profilers are easy to set up but are less accurate as they provide estimates. Among the most popular ones we have: `perf` and `powerstat`. Power Monitors are hardware devices wired directly to the system to profile, e.g., to the battery of the system. Therefore, they are more accurate but also more complex to set up. In this work, we exploit two power monitors for our experiments: the Monsoon [19] and the Watts up Pro? [22]. Instead of reporting total energy usage in joules, several power monitors report the power consumption in watts. They read, at each instant `t`, the current intensity (I) and voltage (V) and calculate the power as $P = I \times V$. The total energy consumption (E) can be derived from the power consumption. When the power consumption is constant, the total energy spent is proportional to the observation interval Δt, that is $E = P \times \Delta t$. As previously mentioned, some power monitors calculate power values querying the system every instant `t` over an observation period Δt. This process results in a dataset, where each row is formed by the power value in watts and the timestamp of the reading. This dataset describes the distribution of power values consumed during Δt. Since power corresponds to the rate at which energy is consumed over time, it is possible to retrieve the total amount of energy spent between two instants t_0 and t_n, by calculating the area bounded by these two instants underneath the distribution. Formally, this area can be calculated by integrating the power consumption values over t_0 and t_n.

2.2 Layered Queuing Network

Layered Queuing Networks (LQNs) are used to describe and analyze the performance of a system [24]. An LQN captures the behavior of a system as a set of interacting entities sending and servicing requests. Incoming requests generate the workload that is handled by system resources such as CPU or Disk. If a request comes to a resource that is already busy, the request is queued. Such a model of computation is peculiar to ordinary queuing models. LQNs extend ordinary queuing models by implementing simultaneous resource possession. Simultaneous resource possession occurs when a resource is blocked waiting for another to finish serving a request. Figure 3 shows the LQN used for analyzing a Train Ticket Booking System. The root task may be used to specify the workload which the system will undergo or receive requests from the environment. In the former case, the task is named reference task and represents the number of users in the system, while in the second case, requests arrive following a rate specified with the λ parameter. System entities are shown as parallelograms and are called tasks. A task provides service through one or multiple entries and has a single host processor. Processors are represented with circles connected to a task and embody system resources. Thus, processors handle the workload generated by the entries. Entries are represented as rectangles within tasks and specify a demand corresponding to the mean time the processor is busy serving the entry. Communication among tasks is described by arrows connecting the entries. These arrows are labeled with the mean number of requests the client task sends to the server task. Carleton University provides a suite of tools including

modeling languages and an analytic solver to create and retrieve performance metrics [7]

3 Our approach

As mentioned in the introduction, we exploit performance models to assist designers in making energy-related decisions. For this purpose, we have conceived a modeling process to reach a good trade-off between abstraction and accuracy of estimates. Developing models of existing systems may help to understand them better, which includes identifying flaws and finding opportunities for improvement. At design time, models are used to describe design choices or to verify conformance to the requirements. Considering that implementation details highly influence the energy consumption of the software, we chose to proceed bottom-up and exploit models for studying existing software systems. This implies the use of models that can be simulated or solved analytically to study "what if" cases and gain fast insights into the system under study. In addition, we see modeling as an opportunity to reduce the complexity and time of measurements. Our approach combines both strengths of measurements and abstraction.

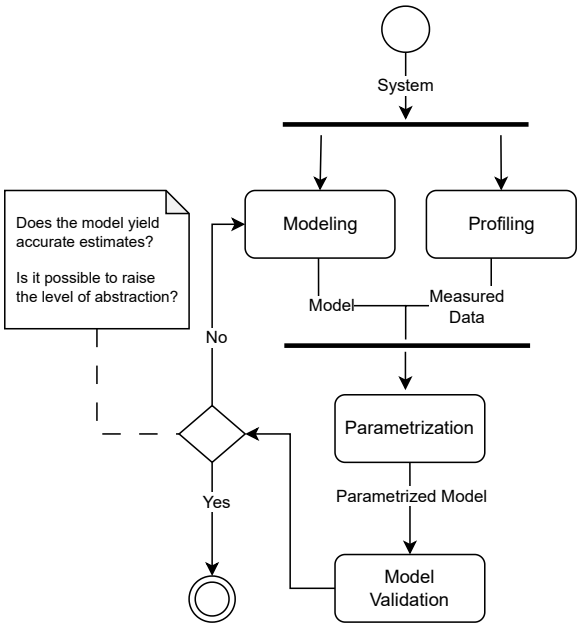


Fig. 1: The process underlying our approach.

Our modeling approach is schematically represented in Figure 1. Each rounded box represents an activity, while labels on the edges embody the exchanged artifacts. Initially, an existing system is modeled and profiled. Profiling means

measuring the characteristics of the system that we planned to evaluate using the models, in this case performance and energy consumption. Modeling and profiling activities can be conducted in parallel. After the end of these two activities, the measured data are used to parametrize the model. Taking performance as an example, we can look at the rate at which requests arrive at the system, or how long it takes software components to handle a request. Parameters such as arrival rate, or service time are the input parameters of the model. Once the model is parametrized, it is validated and possibly refined. In the validation step, the correctness of the model and the accuracy of the estimations are considered. Moreover, during this phase, designers refine the model removing unnecessary details and simplifying it. The process stops when consensus about model accuracy and abstraction level is reached. We envisage that, under a set of assumptions, it is possible to exploit the flexibility of performance models for estimating energy consumption. Briefly speaking, since energy consumption is a metric based on time, we can relate energy consumption and resource utilization over a fixed observation time. Our approach, at the moment, considers only the cases in which *energy consumption grows linearly with execution time*. However, this is not always true. As reported by Cruz et al [9], some mobile architectures have fast but power-hungry CPUs for processing heavy tasks and slower but more efficient CPUs for less time-consuming tasks. In addition, CPUs frequency scaling mechanisms should be considered, as they impact the non-linearity between workload intensity and power consumption [18]. Despite this, the approach has significant benefits since performance models can be utilized to scale workload and retrieve energy consumption values along with resource utilization estimates.

As a result of the modeling process depicted in Figure 1, we obtain a model approximating the behavior of the system with a certain degree of accuracy. This approximation stems from the behavior observed while profiling the system. So, the model reproduces the behavior observed under the experimental setup of the profiling phase. For example, profiling the system under a given workload will produce a model representing resource usage under that specific workload. This aspect becomes even more important when it comes to power consumption. Indeed, behavior and power consumption are heavily connected. Different experimental settings will result in different system behaviors and thus different power distributions. On the other hand, we can exploit the relationship between behavior and power distribution to infer that the modeled behavior will induce a power distribution similar to the system one. Indeed, from the performance model solution we know when resources will be busy handling requests. For this reason, we can map the time interval resources are busy onto the power distribution measured during the profiling phase. Figure 2 shows an example of mapping between model behavior and the power distribution measured on the system. In this example, the power distribution is measured while running several times a software on a server. The blue cross on the x-axis represents the end of a single execution. Requests arrive periodically and require CPU and disk at the same time. The CPU, depicted by the blue area, is occupied for a longer interval than the disk, which is depicted in orange. White areas represent the time interval

when resources are idle. Each colored segment underneath the power distribution represents the energy consumed by a resource during that time interval (see Section 2.1). Therefore, it is possible to obtain the energy consumption of a resource by integrating the areas where the resource was busy (1). Thus, given a resource, the sum of all the intervals in which a resource was busy (i.e., same color intervals in Figure 2) represents the total energy demand (i.e., $ED(res)$) during the observation time (Equation 2). Accordingly, we can derive the average consumption per visit of each resource, namely $E(res)$ (Equation 3). This value is tied to a visit, so it reflects the energy spent serving a specific software task. For example, if the CPU spent 2 seconds serving a visit, the energy spent per visit refers to the consumption during two seconds. So, the energy consumed per visit is measured in $\frac{Joule}{Visit}$. The Joule consumed per second (i.e., $e(res)$) can be obtained from $E(res)$, which is unbound to the size of a software task. We calculated $e(res)$ by dividing $E(res)$ by the average time the resource is busy performing a software task (Equation 4). Since $e(res)$ is decoupled from the size of the task visiting a resource, we can use it to estimate the energy consumed by a resource busy serving a task with whatever size. In other words, since $e(res)$ is measured in $\frac{Joule}{s}$, it is untied from the workload and becomes a property of the resource. Thus, we assume that it does not sensibly vary by scaling the workload. In this way, we can use $e(res)$ as a multiplier of the busy time of a resource and estimate the energy consumption in case of larger workload sizes. This method allows designers to scale up the estimations retrieved during low-effort experiments to predict the energy and performance of more complex scenarios. In this way, designers can very quickly obtain estimates for complex cases and thus avoid laborious and complex experiments.

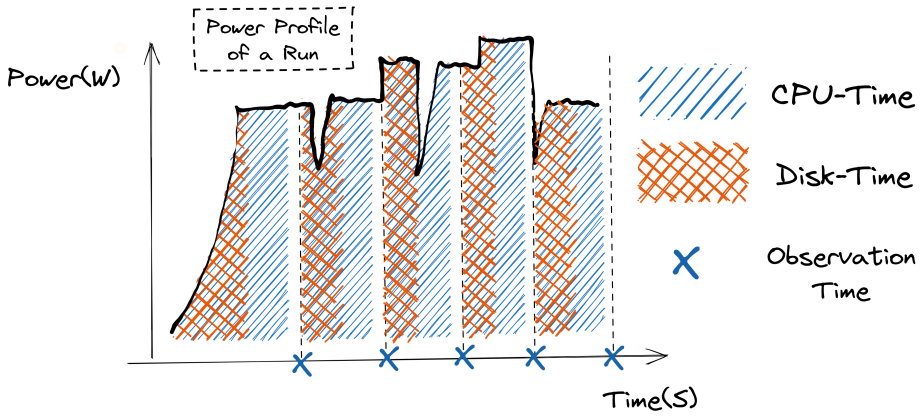


Fig. 2: Sample Power Profile highlighting CPU and Disk busy time, respectively, in blue and orange.

$$E(res, i) = \int_{t0,i}^{S_{res}} P(t)\, dt [\frac{Joule}{Visit}] \tag{1}$$

$$ED(res) = \sum_{i=1}^{\#Visit} \int_{t0,i}^{S_{res,i}} P(t)\, dt [Joule] \tag{2}$$

$$E(res) = \frac{ED(res)}{\#Visit} [\frac{Joule}{Visit}] \tag{3}$$

$$e(res) = \frac{E(res)}{S(res)} [\frac{Joule}{s}] \tag{4}$$

where:

res = a resource
$t0$ = the instant a resource starts to be busy
i = ith visit to a resource
$\#Visit$ = total number of visits to a resource
S_{res} = the average time the resource spends serving a software task

Running example: the Digital Camera

We deployed the application of a Digital Camera (DC) on a BeagleBone Black (BBB) [4] development platform. The BBB is a ARM-based single core platform equipped with Linux Debian, which, in our setting, executes only the services of the operating system and the DC. As our approach envisions (see Section 3), we set an experiment in which the DC is subject to a synthetic workload. While the DC processes the workload, we measure the power required by the BBB using a Monsoon Power Monitor [19], which is placed between the BBB and a notebook. The notebook orchestrates the experiment, stores the power consumption recorded by the power monitor, and records the performance of the BBB. Performance measures include the time DC takes to process an image (i.e., the response time) and resources utilization. The notebook queries the operating system of the BBB and retrieves, for each execution, the busy time of the CPU.

The workload consists of a stream of image batches. A batch contains 30 pictures of the same format chosen between 2K, 4K, and 8K. A total of thirty batches are provided to the application, i.e., 10 per format. The DC processes all images sequentially in a batch, then pauses for two minutes before continuing. Their arrival is randomized to avoid any influence of image size on measurements [23]. The batch size is set to 30 to achieve statistical significance of metrics derived from observed behavior, such as resource utilization.

We determined the energy spent by the CPU to operate a batch of a particular format based on Equation 3. Therefore, using the measured response time, we calculated the e multiplier using Equation 4, which represents the average power in $\frac{Joule}{s}$ spent by the CPU. As Section 3 remarks, e is a property of

the resource, which is detached from the characteristics of the visiting task and therefore has the same value across scaled workloads. If we know or estimate the $S(res)$ variable of Equation 4, we can use the e profiled for the batches of 2K images to discover the average energy spent for batches of format 4K and 8K. Consequently, we obtain estimates of the energy consumed for batches of 4K and 8K images without taking any measurements. We used an LQN to simulate the arrival of 30 images of 4K and 8K and estimate the response time for the corresponding batch. The LQN has a single task, representing the DC, connected to a single processor, which embodies the CPU. Incoming images arrive following a rate (i.e., λ). By varying the λ parameter, we replicated sequential arrival of 4K and 8K images, while changing the service time of the DC task, we set the average CPU processing time per image. In fact, the service time of the CPU differs based on image size. The model yields estimations concerning both the time it takes the CPU to handle a batch of 30 images and CPU usage.

Table 1 provides the results according to the image format of a batch. The table includes the arrival rate λ, the time spent handling a batch, i.e. the response time, CPU utilization, and energy consumed per batch. Columns containing two values show the measured value on the left and the corresponding estimates on the right. The estimates for the energy consumed per batch are calculated multiplying $1.57\frac{J}{s}$, which corresponds to the e calculated for batches of 2K images, by $240.30s$ and $960.60s$, which is the estimated response time for batches of 4K and 8K images. By comparing estimates to the measurements, it can be concluded that the results are promising. Finally, the BBB data sheet reports a range of $1.04\frac{J}{s}$ to $2.3\frac{J}{s}$ consumed by the platform when subject to various load [4]. The $e(CPU)$ multiplier of the DC falls within this range. This observation confirms the reliability of the $e(CPU)$ used for the analysis and the value of the approach for evaluating particular combinations of hardware and software. The small complexity of this case study and the availability of the DC source code, have simplified the mapping process between power distribution and the busy time of the CPU. Indeed, the DC executes only few functions in sequence, and from the source code we could see when the CPU operations were performed. In addition to the plethora of analyses that can be done by varying LQN parameters, we also gain benefits in terms of time. In fact, we obtained energy estimations without performing experiments in the 4K and 8K cases.

4 Evaluation

In light of the promising results obtained from the experiments with the Digital Camera, we decided to validate the method using a more widely used case study: Train Ticket Booking System (TTBS) [14], which is an application comprised of 68 Docker containers that manages bookings of a railway system.[4]

Consistently with the method described in Section 3, we set up a testbed for profiling TTBS and supplied a synthetic workload to the application. Therefore,

[4] For our measurements, we used release 0.0.4: https://github.com/FudanSELab/train-ticket/tree/release-0.0.4

Table 1: Results for the Digital Camera and Train Ticket Booking System according to the size of the input. Legend: IS: Input Size; λ: Arrival Rate; R: Response Time; U: CPU Utilization; e: Average Power Consumption; EC: Average Energy Consumed. Columns with double values indicate measured and estimated values (on the right).

IS	λ ($\frac{images}{s}$)	R (s)	U (%)	e ($\frac{J}{s}$)	EC (J)
Digital Camera					
2K	0.48	60.30 - 60.30	96.30 - 96.48	1.57	95.27 - 95.16
4K	0.12	240.36 - 240.30	96.76 - 96.12	1.59	382.46 - 379.24
8K	0.03	960.73 - 960.60	97.39 - 96.06	1.59	1537.96 - 1516.04
Train Ticket Booking System					
75	6.45	4.09 - 4.63	35.96 - 39.86	78.56	321.99 - 364.17
150	9.17	8.89 - 9.27	50.72 - 56.73	80.89	719.82 - 728.34
225	10.32	13.86 - 13.90	57.88 - 63.77	82.45	1143.19 - 1092.51
300	11.02	18.96 - 18.54	61.70 - 68.10	82.28	1560.54 - 1456.68
375	11.34	24.95 - 23.17	64.91 - 70.08	82.02	2047.20 - 1820.85
450	11.51	29.89 - 27.81	66.32 - 71.13	82.77	2474.40 - 2185.03
500	11.64	33.73 - 30.90	67.67 - 71.94	82.67	2788.76 - 2427.81

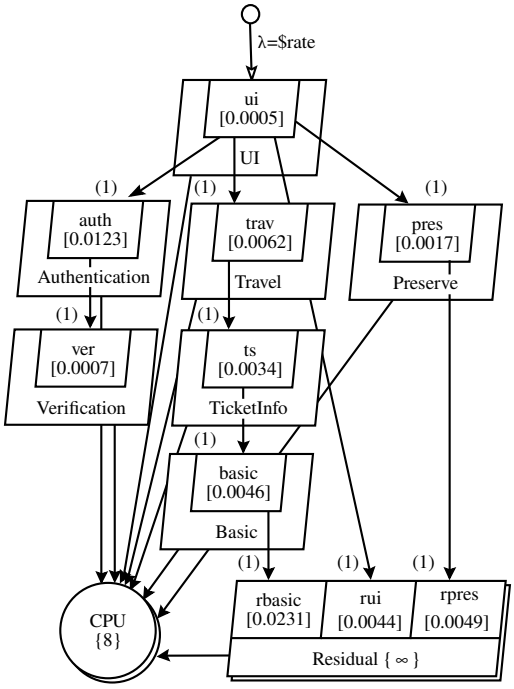
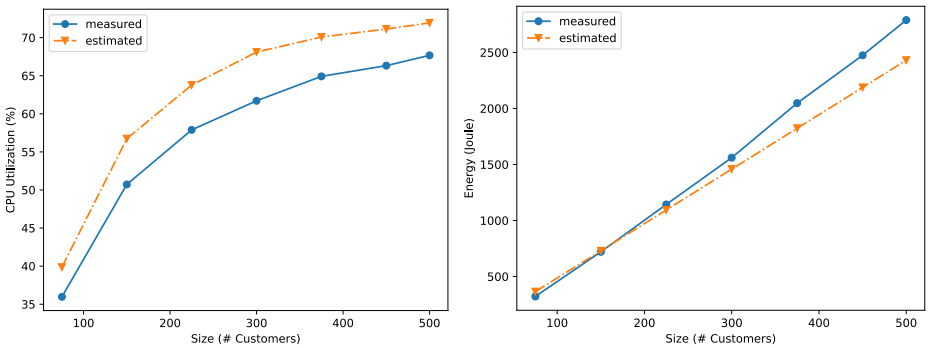


Fig. 3: Layered Queuing Network of Train Ticket Booking System. The variable $\$rate$ varies based on the workload to provide to the model.

we modeled TTBS through an LQN (see Fig. 3) and parametrized the model with the measurements retrieved during the fastest experiment. For the sake of space, we do not show the iterations, described in Section 3, performed to obtain the LQN. Each task in the model embodies a Docker container, which has its service time indicated in square brackets. We remark that this value stands for the average time a container kept the CPU busy. We included a task called `Residual` that models the time spent on the CPU by unrepresented containers. Thus, this task can be seen as a delay that is triggered as requests arrive.

The testbed consists of two machines, M1 and M2, used, respectively, to record the measurements and run TTBS. Both machines run Ubuntu Linux. M2 has 32 GB of RAM and 8 CPUs. It is worth to remark that using two different machines we reduce the perturbation on the machine where the application is running. Hence, we collected results as cleaner as possible. We provide TTBS with a variable-sized burst of customers and measure the performance and the energy expense of M2 for each burst. The bursts can have size equal to 75 (i.e., the one we used to parameterize the LQN model), 150, 225, 300, 375, 450, 500 and were randomly supplied to TTBS for 30 times. This means for each size we collected 30 readings for a total of 210 executions. Randomization is necessary to remove the burst size from the factors that might influence the readings [23]. Further, we inserted a one-minute pause between executions to allow M2 to cool and thus prevent subsequent executions from affecting the profiled data. We empirically validated the pause we need to have a fresh machine. Measurements are coordinated through a bash script running on M1. The script runs JMeter [2], which generates a burst of customers operating on TTBS. At the same time, it records application and M2 performance from the operating system of M2, while the power consumption of TTBS is recorded from a Wattsup Pro power monitor [22] connected to M2. Additionally, we measured disk utilization, but excluded it from the analysis because it was too low to be meaningful.



(a) Performance (b) Energy Consumption

Fig. 4: Comparison between measured and estimated results.

We set the arrival rate, i.e., the λ parameter of the LQN in Figure 3, and the service time of each task according to the data collected while supplying 75-customer bursts. Therefore, we incremented the λ parameter to replicate the arrival of 150, 225, 300, 375, 450, and 500 customers. The LQN returns CPU utilization predictions that we compared against the measurements. Table 1 reports performance and energy metrics for each batch size. It provides the arrival rate, i.e., the one we also supplied to the LQN, the response time, CPU utilization, the average power consumed by the CPU, i.e., the e multiplier, and the average energy consumed per batch. The columns containing two values show the measured value on the left and the corresponding predicted value on the right.

As expected, the CPU utilization rises according to the size of the burst. In our experiment, CPU utilization ranges from 35.96% when supplying a burst of 75 customers to 67.67% with a burst size of 500 customers. Figure 4a shows the distance between CPU utilization estimates and measurements varying burst size. The predictions slightly overestimate the measurements. This overestimation is quantified by the Root Mean Squared Error (RMSE) which equals 5.27%. Moreover, we obtained a Mean Absolute Percentage Error (MAPE) of 9.24%, which confirms the accuracy of the CPU utilization predictions. Besides utilization, the LQN returns response time estimates for each burst size. Therefore, it is possible to estimate the average energy consumed for a given burst by combining the corresponding response time prediction with the e multiplier calculated for the 75-customer burst, i.e., $\frac{E(CPU)}{S(CPU)} = \frac{321.99J}{4.098s} = 78.56\frac{J}{s}$. For example, given the response time estimation for a 500-customer burst, i.e., 30.90 seconds, we calculated the corresponding energy consumption by multiplying it by e using Equation 4. We obtained an estimation of 2427.81 J, which is lower than the measured energy consumption, i.e., 2788.76 J. Figure 4b summarizes energy consumption predictions for each tested burst size. The energy consumption estimates are quite accurate as also evidenced by the RMSE and the MAPE which are equal to 200.16 J and 8.72%, respectively. However, we can observe that as the burst size increases, the difference between the prediction and the measured value also increases. As it can be seen from Figure 4, the RMSE between the energy estimation and the measured one shows a divergence trend meaning that we can suppose having a larger RMSE as the size of the burst grows up. Whereas, the response time trend appears to be convergent. We suppose that this phenomenon may be due to the amount of data collected during the shortest experiment. In our case, this can be noticed by comparing the value of the e across burst sizes. There is a difference of approximately $4\frac{J}{s}$ between the e calculated with 75-customers burst data and the one measured for greater workloads, which is nearly $82\frac{J}{s}$. Finally, by exploiting the LQN, we gain savings in terms of experimentation time. We spent nearly 5 hours collecting all the data for different burst sizes. This period can be reduced by measuring the system undergoing bursts of 75 customers and exploiting the model for predicting energy and performance for greater workloads. By doing so, we would have spent only 35 minutes for experimentation versus 5 hours for measuring the 7 cases.

5 Threats To Validity

This discussion of threats to validity follows the classification made by Wohlin et el. [23]. The results of the study might be affected by *Conclusion Validity* threats due to the low significance of the sample collected during the lower-effort experiment. In fact, due to the short duration of this experiment, the sample collected may not be enough to accurately characterize either the LQN parameters (e.g., service time of containers) or the energy data (e.g., value of $e\frac{J}{s}$). This inaccuracy affects the estimates since we use this dataset to parameterize the LQN and derive energy consumption values for heavier workloads. Moreover, we considered a small set of data points, i.e., 3 image formats for the Digital Camera and 7 different burst sizes for Train Ticket Booking System. This limitation hampers generality and could influence the results, as the linearity between energy consumption and burst size for Train Ticket Booking System. In both case studies, we consider only the load handled by the CPU and scaled workloads. The findings might not be confirmed in situations involving more resources and different types of workload. Therefore, they might not be generalizable and the work might be affected by *Construct Validity* threats. Finally, we do not consider a broad sample of hardware/software systems. For example, we do not examine battery-powered systems, which may have power-saving modes. These characteristics might impact the measurements of energy consumption and performance. As a result, the study might be affected by *External Validity* threats, making it difficult generalize the findings to all types of systems.

6 Related Work

To the best of our knowledge, this is the first study investigating and quantitatively evaluating how performance models (specifically, LQNs) can be exploited to make accurate energy estimations of software systems. Moreover, in our case, the models are used to support measurement-based experiments and reduce experimentation time, thus assessing situations that designers aimed to measure. Several papers in the literature use queuing models to define energy-aware behaviors. These works come from different domains, such as robotics [11], wireless sensor networks (WSNs) [16,17,25], or cloud computing [3]. Cerotti et al. [8] use a queuing model to improve the utilization of the servers in a data center. Indeed, depending on the workload, some servers may be subject to long periods of low utilization, which still generate significant energy consumption. The queuing model incorporates a controller which manages the incoming workload so that servers maximize their throughput and resource utilization. So, each request will require less energy to serve, reducing the total energy consumption. Marsan and Meo [1] apply queuing models to optimize the energy footprint of a university WLAN. The authors consider the areas covered by multiple access points (APs). Co-located APs can be turned on/off, depending on the capacity of the group of APs to provide service and the number of active users accessing the WLAN. This situation is modeled with a queuing system which outputs the number of

APs of a group that should be active to handle a given workload. In some of the situations, the authors manage to save even more than half of the energy usually expended to power the WLAN.

7 Conclusions

In this paper we have introduced a model-based approach for simplifying the energy consumption estimation of software systems. We have exploited the linear dependency of energy consumption and performance to extrapolate estimations of the former one in scenarios that would require high measurement times in practice. We tested the approach using a running example: Digital Camera, then validated the results on a more complex application, Train Ticket Booking System. The experimental results are quite promising, thus we plan to apply our approach to larger-size energy-critical software systems. Besides, we intend to examine the performance and energy consumption of resources other than the CPU, such as the disk and network. Although the approach has been implemented on top of LQN model, the whole process is independent of the modeling notation adopted for sake of performance analysis. Therefore, as further future work, we plan to consider different modeling notations that could be more suitable in specific application domains.

References

1. Ajmone Marsan, M., Meo, M.: Queueing systems to study the energy consumption of a campus WLAN. Computer Networks **66**, 82–93 (Jun 2014). https://doi.org/10.1016/j.comnet.2014.03.012
2. Apache Software Foundation: Apache JMeter. https://jmeter.apache.org, accessed: 2023-04-12
3. Balde, F., Elbiaze, H., Gueye, B.: Greenpod: Leveraging queuing networks for reducing energy consumption in data centers. In: 2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN). pp. 1–8 (2018). https://doi.org/10.1109/ICIN.2018.8401602
4. BeagleBoard.org Foundation: The BeagleBone Black Development Platform. https://beagleboard.org/black, accessed: 2022-11-11
5. Belkhir, L., Elmeligi, A.: Assessing ICT global emissions footprint: Trends to 2040 & recommendations. Journal of Cleaner Production **177**, 448–463 (2018)
6. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice, Second Edition. Synthesis Lectures on Software Engineering, Springer International Publishing (2017)
7. Carleton University Software Performance Research Group: Layered queuing network solver. https://github.com/layeredqueuing, accessed: 2023-03-23
8. Cerotti, D., Gribaudo, M., Piazzolla, P., Pinciroli, R., Serazzi, G.: Multiclass queuing networks models for energy optimization. In: Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools. p. 98–105. VALUETOOLS '14, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, BEL (2014). https://doi.org/10.4108/icst.Valuetools.2014.258214, https://doi.org/10.4108/icst.Valuetools.2014.258214

9. Cruz, L., Abreu, R.: Performance-based guidelines for energy efficient mobile applications. In: 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft). pp. 46–57 (2017). https://doi.org/10.1109/MOBILESoft.2017.19
10. Eder, K., Gallagher, J.P., López-García, P., Muller, H., Banković, Z., Georgiou, K., Haemmerlé, R., Hermenegildo, M.V., Kafle, B., Kerrison, S., Kirkeby, M., Klemen, M., Li, X., Liqat, U., Morse, J., Rhiger, M., Rosendahl, M.: ENTRA: Whole-systems energy transparency. Microprocessors and Microsystems **47**, 278–286 (Nov 2016)
11. Ekren, B.Y., Akpunar, A.: An open queuing network-based tool for performance estimations in a shuttle-based storage and retrieval system. Applied Mathematical Modelling **89**, 1678–1695 (2021). https://doi.org/10.1016/j.apm.2020.07.055
12. Esmaeilzadeh, H., Cao, T., Yang, X., Blackburn, S., McKinley, K.: What is happening to power, performance, and software? IEEE Micro **32**(3), 110–121 (2012). https://doi.org/10.1109/MM.2012.20
13. Franks, G., Al-Omari, T., Woodside, M., Das, O., Derisavi, S.: Enhanced modeling and solution of layered queueing networks. IEEE Transactions on Software Engineering **35**(2), 148–161 (2009). https://doi.org/10.1109/TSE.2008.74
14. Fudan Software Engineering Laboratory: Train Ticket Booking System. https://github.com/FudanSELab/train-ticket, accessed: 2023-04-12
15. Georgiou, K., Xavier-de Souza, S., Eder, K.: The iot energy challenge: A software perspective. IEEE Embedded Systems Letters **10**(3), 53–56 (2018)
16. Ghosh, S., Unnikrishnan, S.: Reduced power consumption in wireless sensor networks using queue based approach. In: 2017 International Conference on Advances in Computing, Communication and Control (ICAC3). pp. 1–5 (2017). https://doi.org/10.1109/ICAC3.2017.8318794
17. Jiang, F.C., Huang, D.C., Wang, K.H.: Design approaches for optimizing power consumption of sensor node with n-policy m/g/1 queuing model. In: Proceedings of the 4th International Conference on Queueing Theory and Network Applications. QTNA '09, Association for Computing Machinery, New York, NY, USA (2009). https://doi.org/10.1145/1626553.1626556
18. Marinescu, D.C.: Cloud computing: theory and practice. Morgan Kaufmann (2022)
19. Monsoon Solutions: Monsoon power monitor. https://www.msoon.com/, accessed: 2021-09-26
20. Tribastone, M., Mayer, P., Wirsing, M.: Performance prediction of service-oriented systems with layered queueing networks. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification, and Validation. pp. 51–65. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
21. Verdecchia, R., Lago, P., Ebert, C., De Vries, C.: Green it and green software. IEEE Software **38**(6), 7–15 (2021)
22. WattsUp: Watts up? pro power monitor. https://github.com/isaaclino/wattsup, accessed: 2023-04-05
23. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29044-2
24. Woodside, M., Franks, G.: Tutorial introduction to layered modeling of software performance (2002)
25. Zhang, Y., Li, W.: Modeling and energy consumption evaluation of a stochastic wireless sensor network. EURASIP Journal on Wireless Communications and Networking **2012**(1), 282 (Sep 2012). https://doi.org/10.1186/1687-1499-2012-282