MEASURE ENERGY CONSUMPTION

Introduction :

- pyRAPL is a software toolkit to measure the energy footprint of a host machine along the execution of a piece of Python code.

- pyRAPL uses the Intel "Running Average Power Limit" (RAPL) technology that estimates power consumption of a CPU. This technology is available on Intel CPU since the Sandy Bridge generation.

More specifically, pyRAPL can measure the energy consumption of the following CPU domains:

- CPU socket package
- DRAM (for server architectures)
- GPU (for client architectures)

Installation :

You can install pyRAPL with pip: pip install pyRAPL

Basic usage :

Here are some basic usages of pyRAPL. Please note that the reported energy consumption is not only the energy consumption of the code you are running. This includes the global energy consumption of all the process running on the machine during this period, thus including the operating system and other applications. That is why we recommend to eliminate any extra programs that may alter the energy consumption of the machine hosting experiments and to keep only the code under measurement (i.e., no extra applications, such as graphical interface, background running task...). This will give the closest measure to the real energy consumption of the measured code.

Decorate a function to measure its energy consumption :

To measure the energy consumed by the machine during the execution of the function foo() run the following code:

```
import pyRAPL

    pyRAPL.setup()

    @pyRAPL.measure
    def foo():
        # Instructions to be evaluated.
    foo()
```

This will print in the console the recorded energy consumption of all the CPU domains during the execution of function foo.

Configure the decorator specifying the device to monitor :

You can easily configure which device and which socket to monitor using the parameters of the pyRAPL.setup function. For example, the following example only monitors the CPU power consumption on the CPU socket 1. By default, pyRAPL monitors all the available devices of the CPU sockets.

```
    import pyRAPL

    pyRAPL.setup(devices=[pyRAPL.Device.PKG], socket_ids=[1])

    @pyRAPL.measure
    def foo():
```

```
    # Instructions to be evaluated.
foo()
```

You can append the device pyRAPL.Device.DRAM to the devices parameter list to monitor RAM device too.

Running the test multiple times :

For short functions, you can configure the number of runs and it will calculate the mean energy consumption of all runs. As an example, if you want to run the evaluation 100 times:

```
import pyRAPL

pyRAPL.setup()


@pyRAPL.measure(number=100)
def foo():
    # Instructions to be evaluated.

for _ in range(100):
    foo()
```

Configure the output of the decorator :

If you want to handle data with different output than the standard one, you can configure the decorator with an Output instance from the pyRAPL.outputs module.

As an example, if you want to write the recorded energy consumption in a .csv file:

```
import pyRAPL


pyRAPL.setup()


csv_output = pyRAPL.outputs.CSVOutput('result.csv')


@pyRAPL.measure(output=csv_output)
def foo():
    # Instructions to be evaluated.


for _ in range(100):
    foo()


csv_output.save()
```

This will produce a csv file of 100 lines. Each line containing the energy consumption recorded during one execution of the function fun. Other predefined Output classes exist to export data to MongoDB and Panda dataframe. You can also create your own Output class (see the documentation)

Measure the energy consumption of a piece of code :

To measure the energy consumed by the machine during the execution of a given piece of code, run the following code :

```
import pyRAPL
```

```
pyRAPL.setup()
meter = pyRAPL.Measurement('bar')
meter.begin()
# ...
# Instructions to be evaluated.
# ...
meter.end()
```

- You can also access the result of the measurements by using the property meter.result, which returns a Result object.

- You can also use an output to handle this results, for example with the .csv output: meter.export(csv_output)

Measure the energy consumption of a block :

pyRAPL allows developers to measure a block of instructions using the keyword with as the example below:

```
import pyRAPL
pyRAPL.setup()

with pyRAPL.Measurement('bar'):
    # ...
    # Instructions to be evaluated.
    # ...
```

This will report the energy consumption of the block. To process the measurements instead of printing them, you can use any Output class that you pass to the Measurement object:

```
import pyRAPL
pyRAPL.setup()


report = pyRAPL.outputs.DataFrameOutput()


with pyRAPL.Measurement('bar',output=report):
    # ...
    # Instructions to be evaluated.
    # ...
report.data.head()
```

Conclusion :

By these processes , we can measure the energy consumption using python programming language in system . Eventhough the application gives many errors based on Operating system ,hardware errors ,errors in measurements , This application is a better for finding values of energy consumption compared to other techniques of identifying the measurements .

Datasets :

Datetime	PJME_MW	PJMW_MW	DAYTON_MW	AEP_MW	COMED_MW	DUQ_MW	PJM_LOAD
########	26498	5077	1596	13478	9970	1458	29309
########	25147	4939	1517	12865	9428	1377	28236
########	24574	4885	1486	12577	9059	1351	27692
########	24393	4857	1469	12517	8817	1336	27596
########	24860	4930	1472	12670	8743	1356	27888

########	26222	5126	1518	13038	8735	1372	29382
########	28702	5493	1598	13692	8993	1402	31373
########	30698	5824	1691	14297	9363	1425	33272
########	31800	5962	1748	14719	9545	1502	34133
########	32359	6019	1825	14941	9676	1556	35232
########	32371	5988	1880	15184	9937	1603	35401
########	31902	5885	1891	15009	10139	1615	35331
########	31126	5764	1874	14808	10326	1617	34582
########	30368	5612	1859	14522	10359	1630	33767
########	29564	5486	1829	14349	10293	1619	33026
########	29098	5474	1807	14107	10240	1613	32620
########	30308	5668	1838	14410	10416	1683	33741
########	34017	6027	1945	15174	11225	1764	36510
########	34195	5973	1968	15261	11907	1786	36783
########	32790	5772	1867	14774	11812	1737	35482
########	31336	5547	1785	14363	11542	1726	34305
########	29887	5358	1711	14045	11149	1669	33277
########	28483	5143	1639	13478	10855	1592	32068
########	27008	4931	1560	12892	10335	1493	30824
########	27526	5340	1713	14097	10126	1484	25013
########	26600	5225	1647	13667	9535	1419	23917
########	26241	5317	1611	13451	9182	1378	23327
########	26213	5310	1608	13379	8945	1359	23286
########	26871	5403	1630	13506	8874	1375	23723

Applications, Energy Consumption, and Measurement

Abstract:

The task of reducing the energy footprint of IT devices and software has been a challenge for Green IT research. Monitoring approaches have primarily focused on measuring the energy consumption of the hardware components of computing devices. The use of applications or software on our computer systems consumes energy and it also affects how various hardware components and system resources consume energy. Consequently, running web browsers applications will utilise considerable energy and battery consumption. In this research, we have run different types of experiments which involve the use of several measuring tools. Firsly, a joulemeter is used to monitor (and measure) the power consumed by the hardware and software while running web-based and stand-alone applications on several devices. Additionally, the tablet in-built battery status checker is used to measure the battery consumption when web-based applications are run on the device.

1 Introduction

Green computing technology focuses on the efficient use of computing resources. In computing devices such as laptops, smartphones, tablets, or other mobile devices, energy consumption is the top priority because they are run on battery, with limited lifespan, as their source of power (Banerjee et al. 2007). With the increasing complexity of IT equipment, the energy consumption rate of these devices system also increases (Silven and Jyrkka, 2007). Most portable mobile device users are conscious of the energy usage by these devices and consequently, they look for ways through which the lifespan of the battery can be extended to serve them longer (Rahmati et al. 2007).

Experiments relating to energy measurement could be at various levels: the hardware level; energy efficiency directive level (Simunic, et al. 2000); operating system (Sagahyroon, 2006); software application or data and user levels (Ravi, et al. 2008). Energy conservation is made possible through the use of different techniques which estimate or forecast energy consumption at the device and application level (Krintz, et al. 2004). The goal of green computing technology is to reduce carbon emission, maximize performance and prolong the lifespan of the computing resources.

1.1 Aim

The aim of this paper is to discuss the results for several investigations conducted on the energy (and battery) consumption for running web-based and standalone applications on Windows and IOS portable computing devices. The following objectives will help to achieve this aim:

- Research Objective 1: To conduct experiments on the measurement of energy consumed for running youtube videos in different web browsers (e.g. Google Chrome, Mozilla Firefox, etc…) on Windows (i.e. laptops), and IOS machines (i.e. tablet);
- Research Objective 2: To conduct experiments on the measurement of energy consumed for playing audio and video files on several media players for windows (on a laptop);
- Research Objective 3: To conduct analyses on data collected in Research Objectives 1 and 2.

This paper will be organised into the following sections: Introduction; Literature Review; Methodology; Results and Discussion; and Conclusion.

2 Literature Review

The Smart2020 report (The Climate Group and GeSI, 2008) predicts an increasing trend of BAU CO2 emissions for the ICT industry. The emissions growth rate for three ICT categories (end-user devices,

telecommunication and networks, and data centers) is expected to decrease from 6.1% 3.8%. By 2020, the ICT industry's footprint is expected to rise to 1.3 $GtCO_2e$ (equivalent to 2.3% of global emissions by 2020). The PC (e.g. desktops, laptops, etc.) footprint (due to its embodied and usage emissions) is the highest (60%) followed by printers (18%), peripherals (13%), smartphones (10%), and tablets (1%). It is estimated that the footprint of end-user devices will grow at 2.3 percent per year to reach 0.67 $GtCO_2e$ in 2020 and thus, energy efficiency improvements in these devices and their proper usage are essential for reducing their overall footprint.

2.1 Energy Consumption of software

Green and sustainable software is a software product that has the smallest possible economic, societal, ecological impact as well as impact on human beings (Ahmed, et al., 2014). This has led to the introduction of various programmes and initiatives that encourages energy efficient software such as green software engineering and Eco-design software (Kaliterre, n.d.).

According to the Greenhouse Gas Protocol (2012), applications are executed with an OS. They affect the power consumption of a device due to data requests and processing. Managing energy requires accurate measurement of the energy available and consumed by a system. This involves monitoring or estimating the resource and energy consumption of hardware and software (Noureddine, et al., 2013). However, a device's power consumption is subjected to the type of application and the task being performed which is evident in our experimental results presented in Section 4 of this paper. In order to reduce the overall power consumption for a web-based or standalone task, it will be necessary to provide users with an insight of the power consumption of the different web-based browser applications (e.g. Google Chrome, Internet Explorer, Mozilla Firefox, Safari, etc…) and also the resource hungry nature of many applications such as movie player and games.

2.2 Energy Consumption of Media Players

Modern technologies incorporate a number of power management features to reduce power waste. Dynamic Voltage and Frequency Scaling (DVFS) can enable the CPU speed to be dynamically varied based on the workload which leads to a reduced power consumption during periods of low utilization (Liu, et al., 2008). The energy-aware dynamic voltage scaling technique has been used to reduce energy consumption in portable media players (Yang & Song, 2009). This scheme showed a relationship between frame size and decoding time. These two cited work merely discuss how energy consumption can be reduced using various techniques, but have not measured the actual amount of energy being consumed by the application. However, the energy consumption of Windows Media Player has been measured using the EEcoMark v2 tool (EecoMark, 2011) but the empirical details of the measurement have not been explicitly discussed. Media playback application power consumption has been analysed by Sabharwal (2011) using windows event tracing. Event tracing does not seem to be an appropriate method for measuring energy consumption because the process itself may have impact on the results. A comparative analysis of energy consumption of media players has been conducted by Techradar (2010). The energy consumption is monitored by playing a DVD on Windows Media Player (WMP) and VLC Media Player. Their research results show that the VLC Media Player is more energy efficient than Windows Media Player. However, the cited work has not mentioned which tool has been used for measurement and additionally, the experiment procedures have not been explitcitly discussed.

2.3 Metrics, Measure and Tools for Energy Consumption

2.3.1 Metrics

Generally, software does not directly consume energy. However, running the software involves the hardware which consumes energy. Therefore, the resource usage metric such as the CPU usage, memory and disk usage are used as the measuring criteria (Mahmoud & Ahmad, 2013). It is important to analyse the energy efficiency of software by observing the amount of resource utilized versus the useful work performed. To measure the power consumption of a system, the power consumed by individual PC components must be measured. Therefore a system wide resource utilization monitoring technique at the user level seems to be more appropriate

2.3.2 Measure and Tools

There is a wide range of methods for measuring the energy consumption of a computer system. Generally, the measurement of energy consumption is grouped into three categories hardware, software and power models (Noureddine, et al., 2013). Measuring energy consumption of hardware using devices such as *wattsup*[1] and the method described by McIntire and colleagues (2007) yields a precise value. *PowerScope* is a tool that uses a multi-meter to measure the energy consumption of applications (Flinn & satyanarayanan, 1999). This method is more precise because it can determine the energy consumption of a specific process and even procedures within the process. However, these methods have some limitations. It can only monitor hardware devices, not flexible, requires additional hardware and the value may fluctuate due to electro-mechanical issues. It is also difficult to upgrade to a more newer and precise monitoring without replacing the entire hardware.

Power models are used to calculate the energy consumption of hardware and software. Kansal and Zhao (2008) use a generic automated tool to profile the energy usage of various resources components used by an application. This method is either too generic or coarse-grained and it is platform dependent (Seo, et al., 2007). The model proposed by Lewis and colleagues (2012) is an integrated model for the calculation of a system's energy consumption.

More promising approaches are software energy measurement using energy application profiler (Noureddine, et al., 2013). In their contribution, Varrol and Heiser (2010) use *Openmoko Neo Freerunner* to decompose the energy consumption of each resource of a system. *PowerAPI* is an Application Programming Interface (API) used for monitoring the real time energy consumption of applications at the granularity of system process (Bourdon, et al., 2013). *PowerAPI* can also be used to estimate the energy consumption of a running process for hardware resources e.g. CPU or for hard disk or for both and many more other resources (Noureddine, et al., 2013). Energy consumption estimation in PowerAPI distinguishes the energy consumption for hardware resources and software blocks of codes.

pTop is a process-level power profiling tool which provides information on the power consumption of the running processes in joules (Do, et al., 2009). It gives the power consumption values for the CPU, computer memory, hard disk and the network interface for each process. The energy consumed by an application is the sum of energy consumed by individual resources in addition to energy consumed by the interaction of these processes (Noureddine, et al., 2013). The windows version also uses the windows API to perform the same task.

Intel energy checker is a software development kit SDK with the capability to provide the Application Programming Interface (API) required to define, measure, and share energy efficiency data (Intel, 2010). The SDK is developed with the intention to facilitate energy efficiency analysis and optimization in data centre and telecoms environment. It can, however, also be used on the client or mobile computing platforms to measure energy consumption (Intel, 2010). There is the functionality of importing and exporting counters from an application, which can measure the time spent for a particular process such as converting a file or reading a video (Noureddine, et al., 2013). However, this approach is limited in flexibility because it requires hardware power meter for power estimation.

Joulemeter is a software tool that can be used to estimate the power consumption of hardware resource and software applications on a computer (Microsoft Research, 2011). It can monitor resources such as the CPU utilization and screen brightness in order to compute these resources energy consumption. It has been used by Vonkoch and colleagues (2011) to measure and compare the power consumption of web browsers. The initial calibration of the *joulemeter* renders it inflexible. However, the power model is easy and straight forward to use. Additionally, it has various versions for different Operating Systems. It is an open-source tool which is available for free downloads from Microsoft. Joulemeter calculates the energy consumption of various components within a computing device: central processing unit (CPU); software application; the monitor etc. (Narayanan, 2005). The

experimental technique is useful for obtaining real-time data and it is cheaper than purchasing an external hardware device because it does not require power supply to operate, there is no need for storage space and disposal after expiry of lifespan (which contribute to e-waste) (Widmer, et al 2005).

3 Methodology

There are three sets of physical experiments conducted to investigate the relationship between various applications and their energy (and battery) consumption. Their respective results are presented in: (i) Section 4.1 – web browser applications and their energy consumption (for a laptop); (ii) Section 4.2 – web browser applications and their battery consumption; (iii) Section 4.3 – media players for Windows and their energy consumption (for a laptop).

3.1. Experiment Setup

Equipment

Hardware devices

The hardware devices used for the 3 sets of experiments have been tabulated in Table 1.

Experiment Set	Device Type	Operating System	CPU	RAM	Screen Size	Experiment Results
1	Toshiba Protégé Z30-A-1D6	Windows 7 Enterprise (64 bit)	4th Gen Intel® Core i7-4600U @2.10GHz 2.70GHz	8.00GB	13.3”	Section 4.1
2	Toshiba satellite L50-1NL	Windows 8.1	4th Gen Intel® Core i3-4005U	4.00GB	15.6”	Section 4.2
3	Apple iPad Air2	IOS 8	triple-core A8X @1.5GHz	2.00GB	9.7”	Section 4.3

Table 1: Experiment hardware devices

Measuring Tool (Software)

Joulemeter 1.2[2] is used for experiment sets 1 and 2 while an inbuilt battery status checker is employed for experiment set 3. There are some specific guidelines which are adhered to when calibrating the Joulemeter. They are: (i) ensure that the laptop battery is fully charged (or more than 50%); (ii) automatic or manual calibration of the energy model. It is necessary to ensure that other applications or programs are not running while the automatic calibration is in progress. Detailed instructions on using the Joulemeter are found in this user manual (Microsoft Research, 2011).

3.2. Experimental Procedures

Experiment Set 1: To investigate the energy consumption of several web browser applications in Windows (the sample interface is shown in Figure 1)

Experimental Steps

i. Create a csv file for saving the real time power consumption data via Joulemeter (by clicking on the *browse button*);
ii. Click on the *start saving button*;
iii. Click on the *start button* to run the application in Google Chrome 1.3.27 (i.e a youtube video[3]);
iv. Click on the *stop saving button* to end the application; v. Repeat the above steps for 9 times;
vi. Repeat all the above steps for each of the following web browser: Internet Explorer 9; Mozilla Firefox 27.0.1; and Safari 5.2.1.

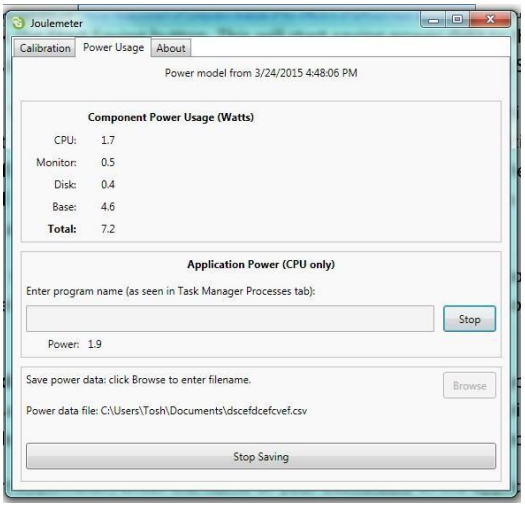

Figure 1: Sample interface for the Joulemeter Measuring Tool The constants of this experiment are:

i. The wifi network used is eudroam;
ii. Constant environment (experiment is carried out within the same office throughout the entire experiment);
iii. The time for all the experiments is from 1200 -1600 for Day 1 and Day 2.

The limitations of the experiments are:

i. The Joulemeter only monitors the energy consumption of the client machine;
ii. Human inconsistency involved when clicking on the essential buttons (see % error due to in Table 6); iii. Technical inconsistency which rendered several of the experiments as errors (Table 2).

Experiment Set 2: To investigate the energy consumption of several web browser applications in IOS

Experimental Steps

i. Record the battery status (in %) of the device manually;
ii. Run a 30 minutes youtube video in the following web browsers: Google Chrome, Safari, Opera Mini and Puffin; iii. Record the battery status (in %) of the device manually immediately after (ii);
iv. Repeat all the above steps for three different times of the day: morning; noon; and night;

Constants in the experiments

i. The experiments are run in the same physical environment; ii. The experiments are run in the same day; iii. Volume of the device is set to *full* while the brightness, *auto*.

Critique of the experiments

i. The number of experiments for each web browser during a certain part of the day ought to be repeated at least 9 times in order to obtain an average of the readings;
ii. The tablet ought to be fully charged (i.e. 100%) before each experiment is conducted.

Experiment Set 3: To investigate the energy consumption of several media players for Windows

Experimental Steps

i. Create a csv file for saving the real time power consumption data via Joulemeter (by clicking on the *browse button*);
ii. Click on the *start saving button*;
iii. Click on the *start button* to run an audio file using KMPlayer in Windows; iv. Click on the *stop saving button* to end the application; v. Repeat the above (i-iv) for a video file;
vi. Repeat all the above (i-v) for the following media players: Windows Media Player (WMP) and VLC Media Player.

Constants in the experiments

i. The device's default settings are used except for the volume which is set to *100%* while the brightness of the monitor, *auto*.

Critique of the experiments

i. The number of experiments for each media player ought to be at least 4 times in order to obtain an average of the readings.

4. Results and Discussion

This section will discuss the results of the data analysis for the three sets of experiments discussed above. The Joulemeter monitored raw data is for the time stamp (in ms), power consumption (in Watts) for each component: CPU, monitor, disk, base and the application. The formula used to calculate the energy consumption by each component is: Energy (J) = Power (W) x Time (s). The results of the calculation for all the experiments runs are shown in Table 2. Note that the data is cleansed so as to omit records with application power consumption = 0W. If the number of remaining records > 50% of the raw data then the cleansed readings of the csv file will be included in the data analysis. However, if it is otherwise, then the experiment is considered an error (see Table2).

4.1 Web Browser Applications for Windows

Browser	Experiment No	Hardware Energy Consumption (J)					Application (J)	Total Energy Consumption (J)	Total Time (s)	Aggregated Average Energy Consumption (J) for t=1s
		CPU (J)	Monitor (J)	Disk (J)	Base (J)	Total Hardware				
Google Chrome 1.3.27.	1	111.168	154.130	0.000	378.319	643.616	106.652	750.268	140.606	
	2	101.205	157.087	0.000	385.576	643.868	96.657	740.524	142.806	
	3	103.981	155.673	0.000	382.107	641.761	98.958	740.719	141.521	
	4	102.365	152.213	0.000	373.613	628.190	98.770	726.960	138.375	
	5	109.315	156.681	0.000	384.580	650.576	105.205	755.781	142.437	
	6	135.710	155.001	0.000	380.457	671.167	126.990	798.157	140.910	
	7**	Error								
	8	126.437	155.297	0.000	381.183	662.917	119.757	782.674	141.179	
	9*	76.769	97.491	0.000	239.296	413.556	74.544	488.100	88.628	
	10	166.749	155.394	0.000	381.421	703.563	157.886	861.449	144.637	5.442
Internet Explorer 9	1	45.764	154.217	0.000	378.532	578.512	38.276	616.788	140.197	
	2	49.357	162.142	0.000	397.985	609.484	43.736	653.220	147.402	
	3	50.285	152.635	0.000	374.649	577.569	44.348	621.917	138.759	
	4	58.794	156.955	0.000	385.252	601.001	50.227	651.228	142.686	
	5	51.618	156.134	0.000	383.238	590.990	46.801	637.791	141.940	
	6**	Error								
	7**	Error								
	8*	32.292	116.732	0.000	286.524	435.548	17.344	452.892	106.120	
	9	47.084	153.871	0.000	377.684	578.640	34.351	612.990	139.883	
	10	57.187	153.783	0.000	377.468	588.438	49.972	638.411	139.803	4.454
Mozilla Firefox 27.0.1	1	128.149	157.489	0.000	386.564	672.203	118.922	791.125	143.172	
	2	82.580	154.375	0.000	378.921	615.876	75.664	691.540	140.341	
	3	97.265	155.539	0.000	381.777	634.581	88.251	722.832	141.399	
	4	73.910	155.802	0.000	382.423	612.134	66.883	679.017	141.638	
	5	87.325	159.672	0.000	391.921	638.918	78.392	717.310	145.156	
	6	128.675	153.585	0.000	376.982	659.243	115.772	775.015	139.623	
	7	87.172	154.356	0.000	378.875	620.403	80.608	701.011	140.324	
	8	86.112	155.170	0.000	380.873	622.155	78.340	700.495	141.064	
	9	98.018	155.813	0.000	382.450	636.281	89.956	726.237	141.648	
	10	90.892	154.529	0.000	379.299	624.720	83.158	707.878	140.481	5.098
Safari 5.7.1	1	102.231	154.069	0.000	389.845	646.145	98.505	744.650	144.387	
	2	88.365	151.310	0.000	383.786	623.461	83.946	707.407	142.143	
	3	98.013	150.006	0.000	381.194	629.213	92.471	721.684	141.183	
	4	92.004	149.867	0.000	380.047	621.917	69.480	691.397	140.758	
	5	93.047	146.667	0.000	371.272	610.985	80.209	691.195	137.508	
	6	154.067	149.436	0.000	380.489	683.992	146.294	830.286	140.922	
	7	95.686	148.821	0.000	378.729	623.236	90.298	713.533	141.378	
	8	104.739	150.848	0.000	383.824	639.410	98.896	738.307	142.157	
	9	97.910	150.878	0.000	383.846	632.633	92.643	725.276	142.165	
	10	94.616	147.126	0.000	372.743	614.485	80.421	694.906	141.267	5.134

Note: * - proportion of the remaining cleansed data > 50% of the raw data and ** is for otherwise and thus considered an error.
White coloured records – results of experiments for Day 1; Blue coloured records – results of experiments for Day 2.

Table 2: The hardware and application energy consumption for running several web browser applications

Table 3 depicts the aggregated data for all the experiments conducted for each web browser: Google Chrome, Internet Explorer, Mozilla Firefox, and Safari. However, in order to provide a fair comparison among the web browsers, the time for running the application will have to be set to 1s (i.e. t = 1s) Consequently, the corresponding energy consumption for each component will have to be normalised for t = 1s (see Table 4). Based on the results shown in Table 4, it seems that Internet Explorer 9 consumes the least energy on laptops, followed by Mozilla Firefox and Safari while Google Chrome seems to be the highest energy consumer. These results are consistent with experiments conducted by the Center for Sustainable Energy Systems at Fraunhofer USA[4], which compare the energy consumption of Internet Explorer 10, Mozilla Firefox and Google Chrome on laptops and desktops. Their results reveal that Google Chrome consumes the highest amount of energy followed by Mozilla

Firefox. The conclusion drawn by them is that Internet Explorer seems to be the most energy efficient web browser.

Browser	Aggregated Average Hardware Energy Consumption (J)					Aggregated Application (J)	Aggregated Total Time (s)	Aggregated Average Energy Consumption for t = 1s (J)
	CPU (J)	Monitor (J)	Disk (J)	Base (J)	Total Hardware			
Google Chrome 1.3.27.5	1033.70	1338.97	0.00	3286.55	5659.21	985.42	1221.10	5.442
Internet Explorer 9	392.38	1206.47	0.00	2961.33	4560.18	325.05	1096.79	4.454
Mozilla Firefox 27.0.1	960.10	1556.33	0.00	3820.08	6336.51	875.95	1414.85	5.098
Safari 5.7.1	1020.68	1499.03	0.00	3805.77	6325.48	933.16	1413.87	5.134

Table 3: Aggregated hardware and application energy consumption for running several web browser applications

Aggregated Energy Consumption for t = 1s	Aggregated CPU (J)	Aggregated Monitor (J)	Aggregated Disk (J)	Aggregated Base (J)	Aggregated Hardware Energy (J)	Aggregated Application (J)	Aggregated Total Energy Consumption (J)	Time (t=1s)
Google Chrome 1.3.27.5	0.85	1.10	0.00	2.69	4.63	0.81	5.442	1.00
Internet Explorer 9	0.36	1.10	0.00	2.70	4.16	0.30	4.454	1.00
Mozilla Firefox 27.0.1	0.68	1.10	0.00	2.70	4.48	0.62	5.098	1.00
Safari 5.7.1	0.72	1.06	0.00	2.69	4.47	0.66	5.134	1.00

Table 4: Normalised hardware and application energy consumption for running several web browser applications (for t = 1s)

Figure 2 is drawn based on the normalised values in Table 4. The normalised aggregated monitor and base energy consumption values seem to be similar for all the web browsers. The CPU energy consumed by Google Chrome seems to be the highest while Internet Explorer seems to be the lowest. On the other hand, the CPU energy consumption for Mozilla Firefox and Safari seems to be similar.

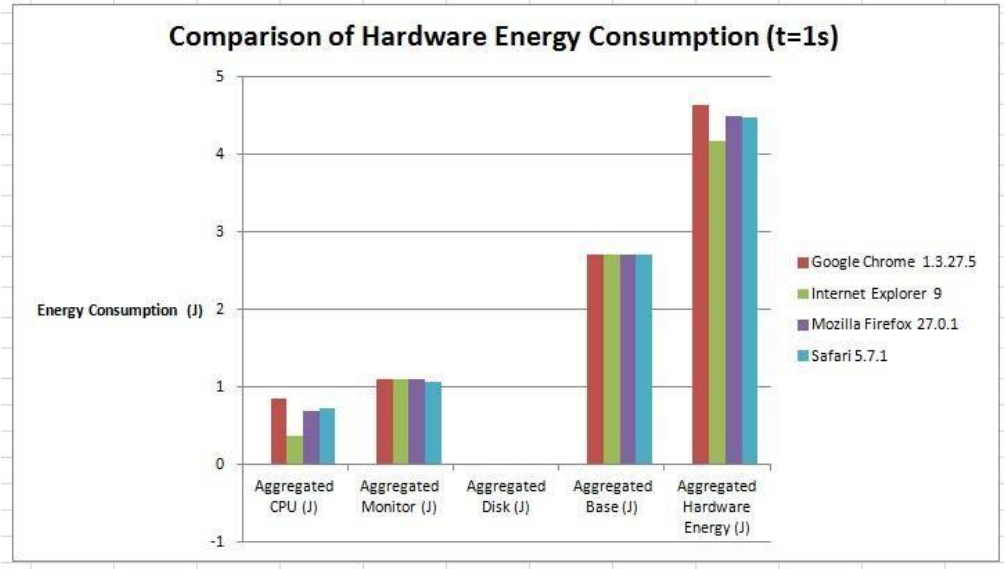

Figure 2: A comparison of the normalised hardware energy consumption for the web browsers

Figure 3 depicts the energy consumption by the hardware and application for each web browser. It can be seen that the energy consumed by the application is very much less compared to the energy consumed by the hardware that runs the application. In order to reduce the overall energy consumption, a further investigation on the interface as well as processes that occur between the software and hardware will have to be conducted and optimized. The energy consumption patterns for the various web browsers are consistent with that for the hardware.

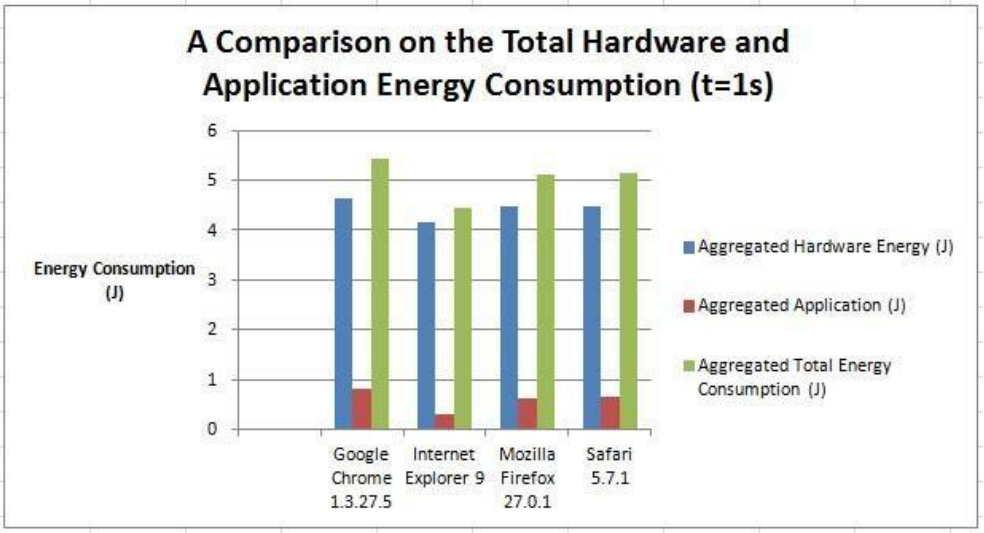

Figure 3: A comparison of the normalised hardware and application energy consumption for the web browsers

Further analyses have been conducted to investigate the ratio of energy consumption between the various web browsers and Internet Explorer (used as a base because it is the lowest energy consumer). Results in Table 5 reveal that the factor for application energy consumption by Google Chrome is almost 3 times that of the Internet Explorer while it is a factor of 2 for Mozilla Firefox and Safari. However, the corresponding factors for the hardware ares lower compared to the factor for the application. Additionally, running a youtube application on Google Chrome (in a laptop) seems to consume approximately 22% more energy than Internet Explorer (note: this is consistent with the 18% finding provided by Center for Sustainable Energy Systems at Fraunhofer USA)[5]. However, the increase for Mozilla Firefox and Safari seems to be in the region of 14%-15%.

Aggregated Energy Consumption for t = 1s	Aggregated CPU (J)	Ratio Comparison to IE
Google Chrome 1.3.27.5	0.85	2.37
Internet Explorer 9	0.36	1.00
Mozilla Firefox 27.0.1	0.68	1.90
Safari 5.7.1	0.72	2.02

Aggregated Energy Consumption for t = 1s	Aggregated Application (J)	Ratio Comparison to IE
Google Chrome 1.3.27.5	0.81	2.72
Internet Explorer 9	0.30	1.00
Mozilla Firefox 27.0.1	0.62	2.09
Safari 5.7.1	0.66	2.23

Aggregated Energy Consumption for t = 1s	Aggregated Hardware Energy (J)	Ratio Comparison to IE
Google Chrome 1.3.27.5	4.63	1.11
Internet Explorer 9	4.16	1.00
Mozilla Firefox 27.0.1	4.48	1.08
Safari 5.7.1	4.47	1.08

Aggregated Energy Consumption for t = 1s	Aggregated Total Energy Consumption	Ratio Comparison to IE
Google Chrome 1.3.27.5	5.44	1.22
Internet Explorer 9	4.45	1.00
Mozilla Firefox 27.0.1	5.10	1.14
Safari 5.7.1	5.13	1.15

Table 5: Energy consumption ratio for the various web browsers

As previously mentioned in Section 3, an experimental error may arise due to human error. The expected total uninterrupted application running time is 2 minutes and 14 seconds (134 seconds). The aggregated expected total application time has been calculated by taking into consideration the proportion of the cleansed data and also the number experiments that have been rendered as errors. The percentages of experimental error for the 4 web browsers are shown in Table 6 and it seems that the experimental error for the Google Chrome is the lowest while the rest is approximately 5 times of Google Chrome. In order to re-affirm the validity of the findings previously discussed, it will be essential to in crease the number of experiments for each browser and measures taken to reduce human inconsistency.

	Time in Joulemeter (s)	Expected Total application time (s)	Experimental error (%)
Google Chrome 1.3.27.5	1221.10	1206.00	1.25
Internet Explorer 9*	1096.79	1035.84	5.88
Mozilla Firefox 27.0.1	1414.85	1340.00	5.59
Safari 5.7.1	1413.87	1340.00	5.51

Table 6: Experimental Error

4.2 Web Browser Applications for IOS

Table 7 depicts the results of running experiments on an Apple ipad Air2 with and IOS operating system. A youtube video has been chosen for this set of experiments and the running time is 30 minutes. Additionally, the experiments are run forthree different times of the day: morning; noon; and night.

Watching YouTube video at (noon)

Browser	Battery % (Before)	Video Duration	Battery % (After)	Battery Consumption %	Brightness	Volume	Date
Chrome	100%	30 (min)	97%	3%	Auto	Full	05/05/2015
Safari	96%	30 (min)	87%	9%	Auto	Full	05/05/2015
Opera mini	87%	30 (min)	78%	9%	Auto	Full	05/05/2015
Puffin	77%	30 (min)	66%	11%	Auto	Full	05/05/2015

Watching YouTube video at (night)

Browser	Battery % (Before)	Video Duration	Battery % (After)	Battery Consumption %	Brightness	Volume	Date
Chrome	66%	30 (min)	62%	4%	Auto	Full	05/05/2015
Safari	62%	30 (min)	58%	4%	Auto	Full	05/05/2015
Opera mini	58%	30 (min)	54%	4%	Auto	Full	05/05/2015
Puffin	54%	30 (min)	48%	6%	Auto	Full	05/05/2015

Watching YouTube video at (morning)

Browser	Battery % (Before)	Video Duration	Battery % (After)	Battery Consumption %	Brightness	Volume	Date
Chrome	47%	30 (min)	40%	7%	Auto	Full	06/05/2015
Safari	39%	30 (min)	33%	6%	Auto	Full	06/05/2015
Opera mini	33%	30 (min)	28%	5%	Auto	Full	06/05/2015
Puffin	28%	30 (min)	18%	10%	Auto	Full	06/05/2015

Table 7: Results of battery consumption for the various web browsers on an IOS device

Firstly, it is noted that the battery consumption by the Puffin web browser is consistently the highest at any part of the day. However, the findings for the rest of the three web browsers (in Table 7) are rather inconclusive. Consequently, the average battery consumption for each web browser has been calculated and shown in Table 8. Once again, the average value for Puffin seems to be the highest while it is the lowest for Google Chrome. However, this finding does not seem to be aligned to the findings in the previous section where the energy consumption for Google Chrome is higher than Safari. In summary, in order to yield more valid results, the following measures (which have been previously mentioned) will have to be taken: conducted repeated experiments for each web browser (at least 9 times in order to obtain the avaerage battery consumption value); use a fully charged battery for each experiment.

Browser	Battery Consumption (%)			Average (%)	Ratio Compared to Chrome
	Morning (%)	Noon (%)	Night (%)		
Chrome	7.00	3.00	4.00	4.67	1.00
Safari	6.00	9.00	4.00	6.33	1.36
Opera mini	5.00	9.00	4.00	6.00	1.29
Puffin	10.00	11.00	6.00	9.00	1.93

Table 8: Average battery consumption for the various web browsers

4.3 Media Players for Windows

To reiterate, the Joulemeter monitored raw data is for the time stamp (in ms), power consumption (in Watts) for each component: CPU, monitor, disk, base and the application. The formula used to calculate the energy consumption by each component is: Energy (J) = Power (W) x Time (s). The results of the calculation for all the experiments runs are shown in Table 9. Normalised data for t = 1s is depicted in Table 10 in order to provide a fair comparison between the various media players.

	Hardware Energy Consumption (J)							
Audio	CPU (J)	Monitor (J)	Disk (J)	Base (J)	Total Hardware Energy Cnsumption (J)	Application (J)	Total Energy Consumption (J)	Time (s)
KMPlayer	1312.99	4593.31	8.38	33228.59	39143.26	433.60	39576.86	2215.24
VLC	1694.42	2870.96	11.36	37110.92	41687.66	1612.72	43300.38	2474.06
WMP	1323.99	15806.66	10.34	37046.85	54187.84	1257.62	55445.45	2469.79
Video								
KMPlayer	2387.37	29492.64	41.48	55298.70	87220.19	2305.68	89525.88	3686.58
VLC	2347.85	29278.87	8.01	54897.88	86532.61	2264.76	88797.37	3659.86
WMP	1906.01	26106.89	161.64	48950.41	77124.95	1124.46	78249.40	3263.36

Table 9: Total hardware and application energy consumption for running several media players in Windows

Audio (t=1s)	CPU (J/s)	Monitor (J/s)	Disk (J/s)	Base (J/s)	Total Hardware Energy Cnsumption (J/s)	Application (J/s)	Total Energy Consumption for t=1s (J)
KMPlayer	0.59	2.07	0.00	15.00	17.67	0.20	17.87
VLC	0.68	1.16	0.00	15.00	16.85	0.65	17.50
WMP	0.54	6.40	0.00	15.00	21.94	0.51	22.45
Video (t=1s)	CPU (J/s)	Monitor (J/s)	Disk (J/s)	Base (J/s)	Total Hardware Energy Cnsumption (J/s)	Application (J/s)	Total Energy Consumption for t=1s (J)
KMPlayer	0.65	8.00	0.01	15.00	23.66	0.63	24.28
VLC	0.64	8.00	0.00	15.00	23.64	0.62	24.26
WMP	0.58	8.00	0.05	15.00	23.63	0.34	23.98

Table 10: Normalised hardware and application energy consumption for running several media players in Windows (for t=1s)

The results revealed in Table 10 suggest that the energy consumption for playing audio and video files seem to be similar for the KMPlayer and VLC Media Player. However, the energy consumption for running the audio file by the Windows Media Player seems to be the highest. However, it is the contrary for running the video file. It could be noted that the audio finding is consistent with Techradar's (2010) finding which shows that the VLC player is more energy efficient than the Windows Media Player. In order to produce more valid results, it is necessary to repeat the entire set of experiments for at least 9 times in order to obtain a more reliable average value. Figures 4 and 5 are plotted based on values in Table 10.

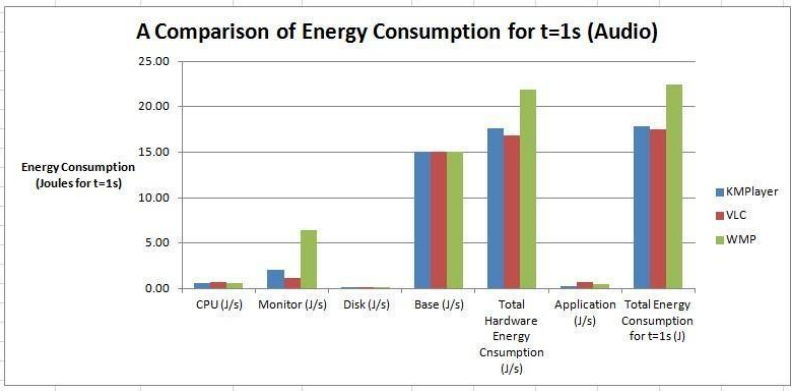

Figure 4: A comparison of energy consumption for playing audio files by several media players in Windows

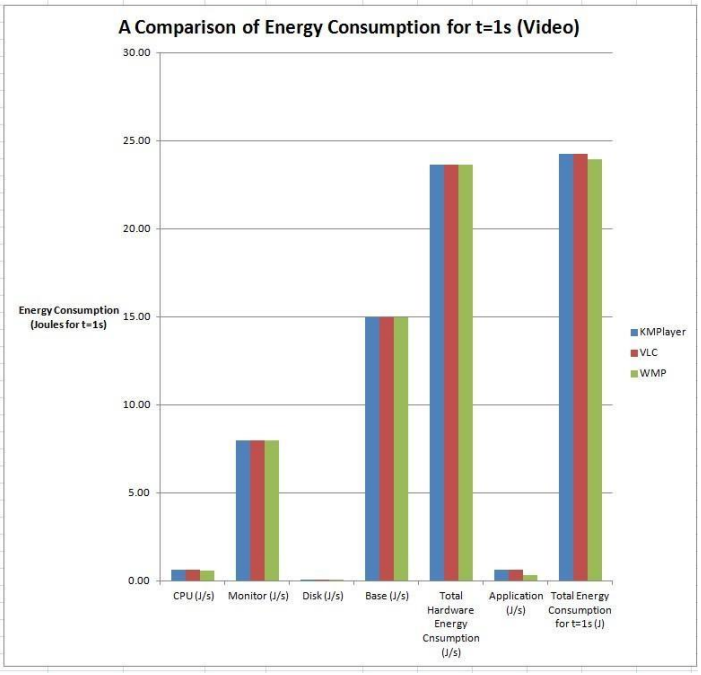

Figure 5: A comparison of energy consumption for playing video files by several media players in Windows

From Figures 4 and 5, it shows that the energy consumption for the disk, CPU and application is very low compared to the monitor and base. The total hardware energy consumed by KMPlayer and VLC for running the audio file seems to be similar while WMP consumes the highest amount of energy. On the other hand, the total hardware energy for running a video file by the three players seems to be similar. The only difference is in the application energy consumption where WMP seems to have consumed the least amount of energy. Further experiments will be necessary to verify this finding.

Conclusion

In this integrated research, we have demonstrated the different tools that could be used to measure the power and battery consumption of various web browser and stand alone applications. The Joulemeter has been employed for the measurement of power consumption by the hardware and software in laptops with Windows operating system while the inbuilt battery status checker has been used to measure the battery consumption in an Apple ipad Air2 with IOS operating system. Some of the results in the experiments (particularly Section 4.1) conducted confirm the findings in existing research. However, further experiments are necessary to verify the findings in Section 4.2 and 4.3 by taking into consideration the experiment critique that have been discussed. This could be completed with additional use of other measure tools that have been discussed in Section 2, and also external measurement devices (e.g. multi-meter, etc…) which would yield more holistic experimental results.

Methodology of Measurement for Energy Consumption of Applications

,tbstract—for IT systems, tntl'lO' awareness can be improved in two ways, (I) in a static or (ii) In a dynamic way. The first way leads lo building tntrizy tfficfont hardware that runs rast and consumes ontr a *r,W* wutl'i. The second wav eonststs or reacting to lnstanta~N>U.'I power eonsumptlon, and or taking decislons that will reduce thi, ccnsumptlon.

In order lo take declslons, it L'I necessary to have a preetse view or the power eonsumptlon or l'llch element or an IT system. Nowadays. II Is only possible lo measure power at the outlet level, or • ,.Ith larger ,rrorts • a the hardware component level, It Is still difficult to estimate the power consumption or slni:le applications running on a computer,

Thls work alms at tnlluuttni: the power consumption or single upplklllions usini: lndirect measurements, Wt describe a methodolo1:y ror pmlictlng the power consumption or a re dtpendlni: on performance counters. and then use these counters to pmlkt the po"er consumptlon or a slni:lt precess,

Kry,.·ords~nergJ' efficiency; process power consumption; performance counters; linear rt,:""5.'llon;

I. INTRODUCTION

Energy awareness can be improved in two ways for IT systems. in a static or in a dynamic way. The static approach is to design energy efficient hardware in the first place. The dynamic approach tries to utilize hardware in an optimal way by moving workload or selling hardware parameters which depend on the workload.

Those two approaches arc complementary. and we mainly address the second one. as is done in autonomous systems. Such systems can react to several events. one of them being power consumption of hosts. Current hardware technology allows only lo measure the power consumption of a whole node. As we manipulate Virtual Machines. we arc interested in the power consumption of each application or VM inside the nodes. To address this problem ii is necessary to create measurement software able to extract the power consumption of each single application.

Several techniques arc currently used. but they either focus on a particular subsystem (such as processor [I]. memory (2). GPU (3)) or are not precise enough (in (4) a comparison of several model leads to an average error of 10%). Generally some process related values are monitored. and based on those values we can then create a mathematical model linking those data to the power consumption of an application.

The contribution of this paper is twofold: first we investi• gate the questions whether the power consumption of a PC can be predicted by measuring OS performance variables. Second, we show how this information can be used to derive the power consumption of a single process.

II. METIIOOOLOGY

Our methodology focuses on standard off-the-shelf PC hardware and software running under the Linux OS. The global methodology is the following:

- We run several applications and synthetic benchmarks
- We log several measurements during the run (including power consumption)
- We use the logged values to derive a mathematical model linking measurements and power consumption

To achieve application power measurement. we use in• direct measurements such as performance counters, process related information (using Linux `pidstat`) and host re• lated information using (using `collectd`). The two first information types are related to a single process. and are to be the base of the produced model. Information based on `collectd` are machine-wide and thus not related to a particular process. But some measurements (such as network traffic) are currently difficult to obtain in a process-related way.

Variables measured using per process performance counters start with *perf_* and include for instance: *perf_task.clt>ck.msecs* (CPU time per second). *perf_cofllext.sll'itches* (context switchcs/s), *perf_CPU.migratim,s* (process rnigrations/s), *perf_pagejall/ls* (page faults/s). *perf_cydes*(CPU cycles/s), *perf_instmctians* (instructlons/s). *perf_cacl,e.rr:fu,mces* (references to the cache's), and *perf_cache.misse.t* (cache misses/s).

Variables measured with `pidstat` start with *pi<!_* and include: *pid_11sr* (Percentage of CPU used by the task while executing at the user level), *pid_system* (Percentage of CPU used by the task while executing at the system (kernel) level), *pid_g1lest* (Percentage of CPU spent by the task in virtual machine), *pid_CPU* (index of the CPU on which is

the process), *pid_minfit_s* (Total number of minor faults the task has made/s), *pid_majfit_s* (Total number of major faults the task has made/s), *pid_kB_rd_s* (Number of kilobytes the task has caused to be read from disk/s), *pid_kB_wr* _s (Number of kilobytes written/s), *pid_kB_ccwr* _s (Number of kilobytes whose writing to disk has been cancelled by the task), *pid_ VSZ* (Virtual Size: The virtual memory usage of entire task in kilobytes), *pid_RSS* (Resident Set Size: The non-swapped physical memory used by the task in kilobytes), *pid_MEM* (Percentage of memory used (by resident memory)).

The system wide metrics from collectd start with *host_* and include: *host_df_df root_used* (Instantaneous used octets on /root), *host_df_df rootJree* (Instantaneous free octets on /root), *host_memory _memory. cached_ value* (Instantaneous cached memory), *host_interface_if_octets. ethl _rx* (Number of octets received during the last second), *host_interface_if_octets.ethl_tx* (Number of octets sent during the last second), *host_interface_if_packets.ethl_rx* (Number of packets received during the last second), *host_interface_if _packets.ethl _tx* (Number of packets sent during the last second), *host_processes_ps_state.blocked_value* (instantaneous number of processes in blocked state), *host_disk.sda_disk_time_write* (average time a write.operation took to complete), *host_cpu.X_cpu.system_value* (processes executing in kernel mode), *host_cpu.X_cpu.wait_value* (waiting for I/0 to complete), *host_load_load_shortterm* (Instantaneous shortterm load, average over lmin), *host_load_load_midterm* (Instantaneous midterm load, average over 5min), *host_irq_irq.X_value* (Number of irqs during the last second), *host_disk.sda_disk_merged_read* (Number of reads that could be merged into other, already queued operations), *host_disk.sda_disk_merged_write* (Idem for write).

A. Monitoring

The currently implemented framework uses two elements, a monitoring part and a synthetic workload part. The monitoring part is split between two computers in order to reduce the impact of measurements on the experiment. A first computer runs the application to be measured and carries out the process and machine related measurements. A second computer is linked to a watt meter and logs the power consumption. At start the two computer clocks are synchronized.

The current limit of the framework is that it cannot follow at the same time a process and its children. So it can currently only be used for applications that do not fork. As most benchmarks actually fork, we had to develop new ones to create the large datasets from which to create the models.

B. Synthetic workload

We created several benchmarks linked to synthetic workloads: Memory, CPU, network, disk and mixed. General behavior of each of them is always the same: They start at 100% of a particular resource and go down to 0% by step of 1%. Each step is 20s. Each second we measure around 200 values (using pidstat, perf counters, collectd and a wattmeter). Thus one experiment produces around 2 MB of data. Starting at 100% and going down to 0% allows to reduce the impact of allocating resources as advised by SpecPower.

III. DATA ANALYSIS

The measured data consists of one response variable (power), and 165 explanatory variables. The task of model building is to explain the response as a function of the explanatory variables. In order to cope with higher order dependencies we also include the squared explanatory variables (marked by a "SQ" at the variable name end), yielding a total of 330 explanatory variables. We did not include interactions between the variables, since this would have lead to an explosion of the number of variables being uncomputable.

The task was now to identify a small number of variables that would explain each individual data set, and additionally a model that would explain all datasets in parallel. The desired models should be parsimonious (number of explanatory variables should be low) and, in relation to this number, the model quality should be as good as possible. Model quality is mainly measured by R^2, the squared correlation coefficient, which explains how much better our model is with respect to the simple data average. In more detail, R^2 measures how much of the total variance (when using the average of the data) is explained when exchanging the overall average with the respective model. Since $0 < R^2 < 1$, and a larger value is better, as an initial goal we wanted to at least achieve $R^2 \geq 0.9$, and consider a model to be "good" if $R^2 \geq 0.95$.

A. Model Quality

Model quality is also measured as average error σ', i.e., the square root of the sum of the squared differences between predicted and measured values. This σ', being the average prediction error, is also set in relation to the average power, yielding the average error in %. Here it must be noted that similar approaches usually report an average error of 5% or higher. Our task thus was to derived models that clearly result in much smaller average errors. However, error in % is of course very much influenced by the mean power consumption of the computers, i.e., higher average power consumption results in a lower error % without yielding a better model, and thus R^2 has more meaning that error in %.

Data was preprocessed by removing data items with at least one NIA value in any variable. The number of removed data lines however is quite small, in the order of below one per mill.

From data analysis we saw that there were only very few extremely influential points which can be identified using Cook's distance [5]. When removing them, the model quality quickly increases. We therefore also designed a simple outlier test removing those points which are at least 4σ away from the mean. This limit must be regarded as being extremely conservative, since usually either 2σ or 3σ are used. The result is a small number of outliers being removed, and we also state R^2 and σ for these cases, as well as the number of removed outliers.

Another test was done to see whether the residuals are normally distributed. However, since the dataset sizes are very high, and there are certain regularities left, using the Shapiro-Wilk test we cannot conclude that the residuals are indeed normally distributed. However, we plotted the data also on quantile-quantile-plots (Q-Q-Plots). On such a plot we put the quantiles of the theoretical normal distribution on the x-axis, and the empirically measured quantiles of the data on the y-axis. If the resulting plot is nearly linear (a line) then we can conclude that the data follows a normal distribution. The Q-Q-plots of our residuals indeed mainly follow a straight line, but show deviances at the tails, which explains why the Shapiro-Wilk test failed.

B. Additive Model Update

In our analysis we start with an empty model in which we only use the average to describe the response. We then use exhaustive search to find the best combination of N variables explaining the response. Since, when having K explanatory variables, the effort for doing this is

$$O(K(K - 1) \cdots (K - N + 1)),$$

this is not possible for larger values of both K and N due to an exponential runtime explosion.

To further reduce complexity we actually chose a mixture approach between additive and subtractive, and first start with a full model containing all variables. Then we remove all those variables that do not have any influence onto the result, by demanding that the p-value (of a t-test testing whether the coefficient is different from zero) of a variable should be larger than 0.5. This reduced K, thus speeding up the following additive approach significantly. This reduction of course is not possible if a full model cannot be created due to a lack of data. In these cases, we had to run the search for $K = 330$, which resulted in a drastically increased runtime. Still, combinations of more than $N = 4$ hardly make sense since they demand run times of days or months. However, there is seldom demand for doing so, since quite often only little can be gained by using more than three variables.

A further possibility is to sequentially add the next best L variables to the model. This means that we can start with an empty model, and then continuously find the best combination of L variables to add to the model. This approach still increases exponentially with L, but only linearly with N. We therefore also state the resulting models for $L = 1$.

IV. Analysis Results

The data consists of the six datasets "Burn CPU", "Mem Loop", "Network", "Tar Kernel", "Disk Read" and "Disk Write". The sizes of these sets are quite large with the exception of "Tar Kernel". The datasets do differ significantly with respect to their power consumption. The empirical cumulative distribution functions (ECDFs) are shown in Figure 1. An ECDF is the integral of the empirically measured density of a dataset. Data is concentrated at those values where the ECDF rises steeply. The datasets influenced by CPU and memory show a rather broad spectrum of power values. Those making heavily use of IIO like "Network" and "Disk X" show a rather narrow spectrum. "Tar Kernel", making use of both CPU and IIO is also narrow, but with a significant shift to the right.

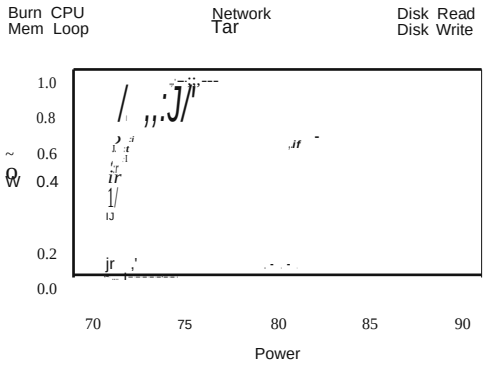

Figure 1. Empirical CDFs (ECDFs) of the respective power consumptions.

In the following we show results of the regression analysis, which also includes plots of the respective residuals. In these plots the x-axis is as important as the y-axis, since it shows the range of the power measurements. Residual sizes must also be related to these ranges.

A. Burn CPU

The dataset "Burn CPU" consists of 2074 complete data records (without NIA values). Both datasets "Burn CPU" and "Mem Loop" are similar in the sense that power can be explained by using one variable only. Table I shows the best variables explaining the power, with outliers taken into account, and when removing outliers. It must be noted that it is not clear why e.g. host_df_df.root_free_SQ yields such good results. There are some other host related variables also yielding high correlations. However, for instance the variables perf_instructions and perf_cycles do make sense

in a pure CPU stress test and also well explain the power consumption.

Variable	R^2	a	%	#Out
host_df_df.root_free_SQ	0.992	0.216	0.289	0
	0.996	0.146	0.195	12
host_df_df.root_used	0.992	0.216	0.289	0
	0.996	0.146	0.195	12
perf_instructions	0.991	0.232	0.310	0
	0.996	0.143	0.191	12
perf_cycles	0.991	0.233	0.310	0
	0.997	0.144	0.192	12

Table I
BURN CPU.

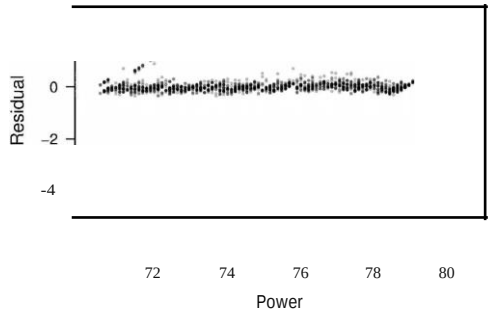

Figure 2. Burn CPU residuals for variable host_df_df.root_free_SQ.

Figure 2 shows the model residuals when using a model with only the variable host_df_df.root_free_SQ as single explanatory variable. There are only a few outliers, nevertheless influencing a significantly.

B. Memory Loop

The dataset "Mem Loop" consists of 2017 complete data records (without *NIA* values). Table II shows the best variables explaining the power,

Variable	R^2	a	%	#Out
perf_context.switches	0.998	0.268	0.339	0
	0.999	0.213	0.270	18
perf_cache.references	0.998	0.285	0.361	0
	0.999	0.196	0.248	25
perf_context.switches_SQ	0.977	0.919	1.163	0

Table II
MEMORY LOOP.

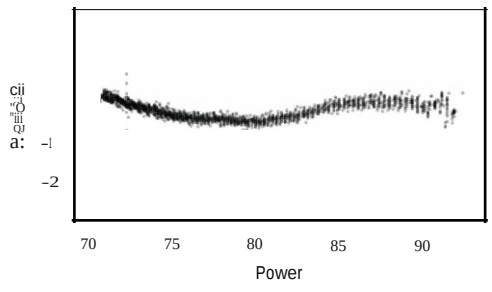

Figure 3. Loop memory residuals for variable perf_context.switches.

Figure 3 shows the model residuals when using a model with only the variable perf_context.switches as single explanatory variable. Again a few outliers are visible, which however do not significantly influence the results. Also, it is obvious that the residui do exhibit a deterministic component. In fact this deterministic dependence could be exploited by further refining the regression model, e.g., by adding a sin()-like function. It turns out that this is not really necessary since the one-variable model is already good enough such that any model extension is superfluous.

C. Network

The dataset "Network" consists of 1937 complete data records (without *NIA* values). Table II shows the best variables explaining the power,

Variable	R^2	a	%	#Out
host_interface_if_octets.eth 1_tx	0.954	0.178	0.247	0
	0.983	0.106	0.147	13
host_interface_if_packets.eth 1_rx	0.953	0.181	0.251	0
	0.980	0.117	0.163	13
host_interface_if_packets.eth 1_tx	0.951	0.183	0.255	0
	0.981	0.114	0.159	13
perf_cycles +	0.967	0.152	0.211	0
host_interface_if_packets.eth 1_tx	0.991	0.079	0.110	16
perf_instructions +	0.967	0.152	0.211	0
host_interface_if_packets.eth 1_tx	0.991	0.079	0.110	16
perf_cycles +	0.967	0.152	0.211	0
host_interface_if_octets.eth 1_tx	0.991	0.077	0.108	17

Table III
NETWORK.

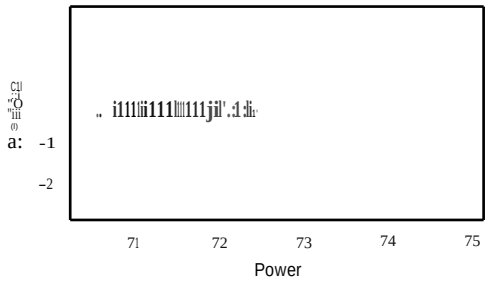

Figure 4. Network residuals for variable host_interface_if_octets.eth1_tx.

Figure 4 shows the model residuals when using a model with only the variable host_interface_if_octets.eth1_tx as single explanatory variable. This variable makes sense when keeping in mind that the workload mainly stresses the network card. Furthermore, Table III and Figure 4 show that although two variables are enough to explain the power consumption, there are a few outliers that do have a significant influence in the result. Removing only 13 out of 1937 data records increases R^2 to 0.98 for one, and 0.99 for two variables.

D. Tar Kernel

This dataset contains only 18 complete data records. Inasmuch it was not possible to reduce the number of

explanatory variables by using a full model first, and thus, searching for optimal variable combinations had to include all 330 explanatory variables (K=330). This was partially compensated by the fact that regression for a small dataset is much faster. Nevertheless, the resulting run times were much higher. Table IV and Figure 5 show the regression results. It can be seen that mainly host oriented variables perform best, e.g., number of blocked processes, disk writing load, but also the load averages.

Variable	R2		%
host_processes_ps_state. blocked_ value	0.911	0.661	0.816
host_processes_ps_state. blocked_ value_SQ	0911	0.660	0.816
host_disk.sda_disk_time_ write_SQ	0911	0.661	0.817
host_cpu2_cpu. wait_ value_SQ + host_load_load_midterm_SQ	0.984	0.290	0.358
pid_kB_rd_s_SQ + host_load_load_midterm_SQ	0983	0.295	0.364
host_cpu2_cpu. wait_ value_SQ + host_load_load_shortterm_SQ	0983	0.300	0.370

Table IV
TAR KERNEL.

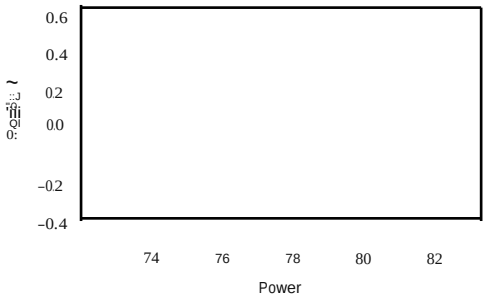

Figure 5. Tar kernel residuals for variables host_cpu.2_cpu.wait_value_SQ + host_load_load_midterm_SQ.

E. Disk Read

This dataset contains 2018 complete data records. As can be seen in Tables V and VI, and Figure 6 there are a few outliers that have a significant impact on the model accuracy. Removing them results in a sufficient fit though. We also computed models for the optimal triple of variables, which however did not improve the model quality significantly. We thus omitted them in Table V. Interestingly single variables do not include disk related variables, but when sequentially adding variables, pid_kB_rd_s_SQ turns up at place third, so it is definitely related to the power consumption.

Table VI shows that for six variables R2 reaches up to 0.98 without outliers, more variables though do not improve the accuracy any more.

F. Disk Write

This dataset contains again 2018 non-empty data records. Similar to "Disk Read", this dataset also suffers from a handful of severe outliers. Figure 7 and Tables VII and VIII show the details. Here it can be seen that power for this workload can be best described without using disk related variables. In general the models perform a little worse for disk related workload (read and write), but when adding more variables the model quality quickly rises significantly.

Variable	R2		%	#Out
perf_ cache. misses	0.824	0.698	0.964	0
	0.956	0.337	0.465	18
perf_context. switches_SQ	0.791	0.761	1.05	0
	0.946	0.361	0.499	22
perf_cache. references_SQ	0.768	0.803	1.109	0
	0.963	0.298	0.412	23
perf_cache.references + pid_system_SQ	0.941	0.405	0.559	0
	0.972	0.277	0.382	12
perf_cycles + pid_system_SQ	0.939	0.410	0.566	0
	0.971	0.281	0.388	12
perf_cache. references_SQ + pid_system_SQ	0.939	0.412	0.569	0
	0.971	0.283	0.391	12

Table V
DISK READ.

Variable	R2		%	#Out
perf_cache.misses	0.824	0.698	0.964	0
	0.956	0.337	0.465	18
perf_cache.misses_SQ	0.909	0.502	0.694	0
	0.956	0.337	0.465	17
pid_kB _rd_s_SQ	0.919	0.474	0.655	0
	0.970	0.280	0.387	18
pid_system_SQ	0.937	0.417	0.576	0
	0.971	0.274	0.379	19
perf_cachereferences_SQ	0.949	0.377	0.520	0
	0.979	0.236	0.326	14
perf_instructions_SQ	0.951	0.367	0.508	0
	0.980	0.231	0.320	14

Table VI
DISK READ, SEQUENTIALLY ADDING SINGLE VARIABLES.

G. *All Datasets*

The previous sections show that our approach yields good results for individual types of work load. In this subsection we derive models for a dataset containing all previously analyzed datasets. Since the datasets are quite heterogeneous it can be assumed that only 1,2 or 3 variables will not yield good models. However, Table IX shows that three variables already yield a good model. The main result however is shown in Tables X and XI, showing nine variables that have been added sequentially. As can be seen, even without outlier removal, these variables explain 99% of the total data variance, yielding them to be excellent predictors of the "All" dataset.

The sequential model shown in Table X also shows the coefficient of variable perf_cache.misses_SQ, which is in the order of magnitude of 10^{-14}. This is explained by the fact that the maximum of this variable (squared number of cache misses per second) is also in the order of magnitude of 10^{14}.

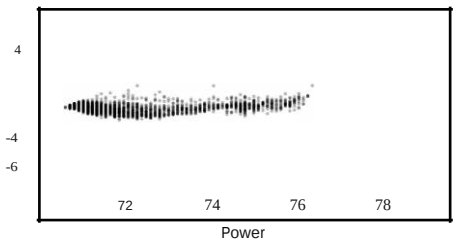

Figure 6. Disk read residuals for variables perf_cache.references + pid_system_SQ.

Variable	R2	a	%	#Out
perf_cache.misses	0.768	0.588	0.822	0
	0.925	0.297	0.415	19
perf_context.switches_SQ	0.635	0.738	1.032	0
	0.955	0.211	0.295	22
perf_context.switches	0.628	0.745	1.0414	0
	0.919	0.281	0.393	23
perf_cycles_SQ +	0.895	0.396	0.553	0
host_cpu.0_cpu. system_ value_SQ	0.967	0.216	0.303	12
perf_instructions_SQ +	0.895	0.397	0.555	0
host_cpu.O_cpu. system_ value_SQ	0.967	0.217	0.303	12
perf_context. switches_SQ +	0.893	0.400	0.560	0
host_cpu.O_cpu. system_ value_SQ	0.970	0.202	0.283	14
perf_context.switches +	0.901	0.385	0.539	0
perf_instructions_SQ +	0.969	0.211	0.295	10
host_cpu. O_cpu.system_ value_SQ				
perf_context.switches +	0.901	0.385	0.539	0
perf_cycles_SQ +	0.969	0.211	0.296	10
host_cpu. 0_cpu.system_ value_SQ				
perf_cache.misses +	0.900	0.386	0.539	0
perf_instructions_SQ +	0.970	0.203	0.283	16
host_cpu. 0_cpu.system_ value_SQ				

Table VII
DISK WRITE.

Figure 8 shows the model residuals when using nine variables. Residuals are taken from the data sets in the order "Burn CPU", "Mem Loop", "Network", "Tar Kernel", "Disk Read" and "Disk Write". As can be seen residuals are not purely random, but also depend on the data set, but not to a high degree. This is further investigated in Section IV-H.

Figure 9 show how the models improve when adding more and more variables sequentially, i.e., the ECDFs of the residuals when using 1, 3, 5, 7, or 9 variables added sequentially. It can clearly be seen that the 9-variables model is superior and yields excellent results.

Finally, Figure 10 shows the quantiles of the residuals plotted against the theoretical quantiles of normal distri• butions. The more straight lines the curves are, the more

Variable	R2	a	%	#Out
perf_cache.misses	0.761	0.588	0.822	0
	0.925	0.297	0.415	19
perf_cache.misses_SQ	0.816	0.524	0.732	0
	0.919	0.308	0.430	20
host_cpu. 0_cpu.system_ value_SQ	0.865	0.449	0.627	0
	0.932	0.296	0.414	21
perf_context.switches_SQ	0.906	0.374	0.523	0
	0.973	0.196	0.274	13

Table VIII
DISK WRITE, SEQUENTIALLY ADDING SINGLE VARIABLES.

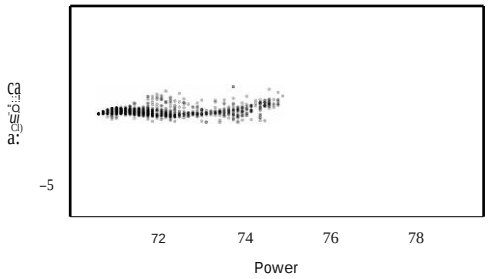

Figure 7. Disk write residuals for variables perf_context.switches + perf_instructions_SQ + host_cpu.O_cpu.system_ value_SQ.

they follow a normal distribution. As can be seen, the model using the optimal triple (03), as well as the model using 9 variables show a large region between -3.5 and 2.5 where they follow a normal distribution nicely. Outside this region, both models show differences in their tails caused by outliers.

H. Global Model Applied to all Datasets

Table XI shows how a model with 9 variable explains all data sets in total. The question now is how this global

Variable	R2	o	%	#Out
perf_cache.references_SQ	0.770	2.003	2.708	0
perf_cache.misses_SQ	0.767	2.017	2.728	0
	0.773	1.987	2.686	16
perf_cache.references	0.683	2.353	3.181	0
pid_usr +	0.920	1.181	1.597	0
perf_cache.references_SQ	0.937	1.046	1.414	57
perf_instructions_SQ +	0.917	1.207	1.632	0
perf_cache.references_SQ	0.933	1.080	1.460	57
perf_task.clock.msecs_SQ +	0.916	1.208	1.633	0
perf_cache.references_SQ	0.933	1.078	1.457	57
perf_cache.references +	0.963	0.807	1.091	0
perf_instructions_SQ + pid_RSS_SQ	0.976	0.646	0.874	62
perf_cache.references +	0.963	0.807	1.091	0
perf_instructions_SQ + pid_MEM_SQ	0.976	0.646	0.874	62
perf_cache.references +	0.963	0.807	1.091	0
perf_instructions_SQ + pid_VSZ_SQ	0.976	0.646	0.874	62

Table IX
ALL DATASETS.

Measuring and Analyzing Energy Consumption of the Data Center

Introduction

The evolution of cloud computing which provides on-demand provisioning of elastic resources with pay-as-you-go model has transformed the Information and Communication Technology (ICT) industry. Over the last few years, large enterprises and government organizations have migrated their data and mission-critical workloads into the cloud. As we are moving towards the fifth generation of cellular communication systems (5G), Mobile Network Operators (MNO) need to address the increasing demand for more bandwidth and critical latency applications. Thus, they leverage the capabilities of cloud computing and run their network elements into distributed cloud resources.

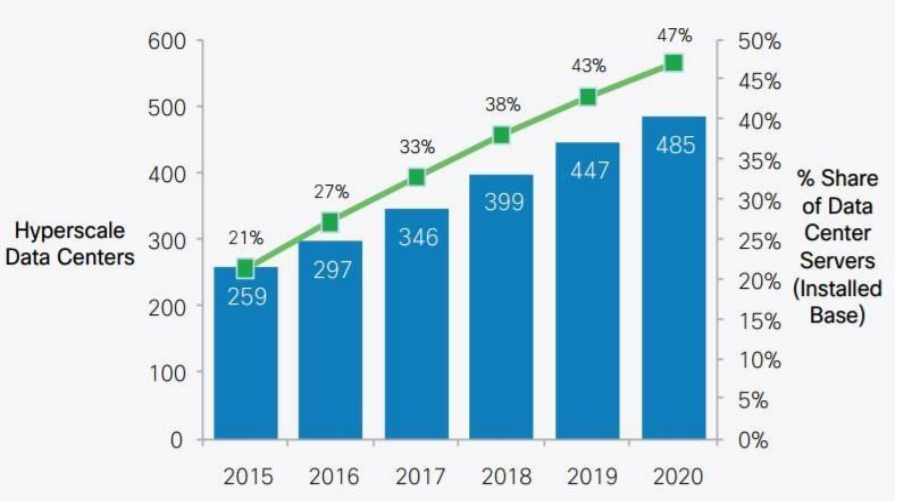

Fig. 1 Growth of hyperscale data centers by 2020 [1].

The adoption of cloud computing by many industries has resulted in the establishment of humongous data centers around the world containing thousands of servers and network equipment. Data centers are large-scale physical infrastructures that provide computing resources, network and storage facilities. Cloud computing is expanding across different industries and along with it the footprint of data center facilities which host the

infrastructure and run the services is growing. Since 2015 there has been 259 hyperscale data centers around the globe, and by 2020 this number will grow to 485 as shown in Fig. 1. These type of data centers will roughly accommodate 50% of the servers installed in all the distributed data centers worldwide [5].

Data centers are promoted as a key enabler for the fast-growing Information Technology (IT) industry, resulted a global market size of 152 billion US dollars by 2016 [2]. Due to the big amount of equipment and heavy processing workloads, they consume huge amount of electricity resulting in high operational costs and carbon dioxide (CO_2) emissions to the environment. In 2010, the electricity usage from data centers was estimated between 1.1% and 1.5% of the total worldwide usage, while in the US the respective ratio was higher. Data centers in US consumed 1.7% to 2.2% of the whole US electrical usage [3]. Fig. 2 shows that over the past few years, data center's energy consumption increases exponentially.

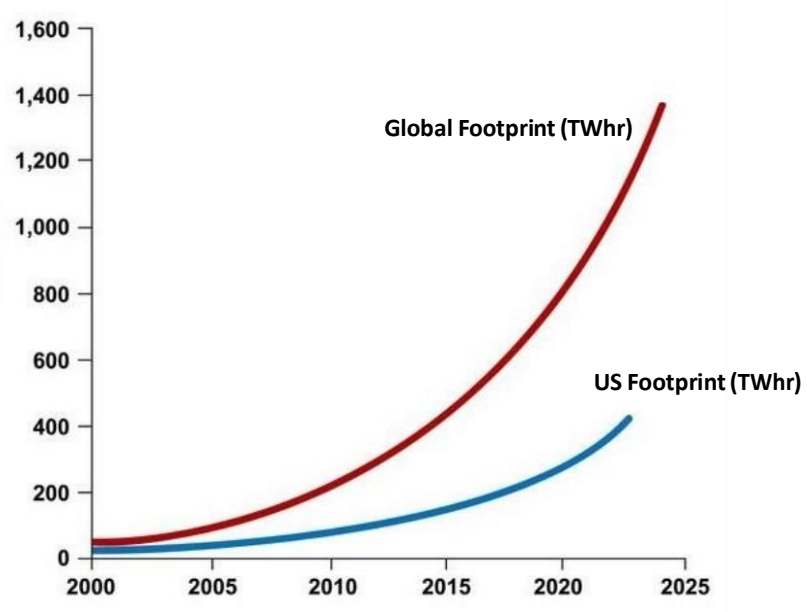

Fig. 2 Projection of data centers' electricity usage [4].

1.1. Objective of the thesis

As discussed in the previous section, the number of data centers increases and so does their respective electricity usage. There has been increased interest from data center's vendors and operators as well as from the academia, to understand how the energy is

consumed among different parts of the data center's infrastructure. A wide body of the research is focused on modeling and prediction of energy consumption. The majority of studies [2] is focused on the energy consumption of the computing subsystems such as servers, network plane and storage, while there are also studies on the energy usage from mechanical and electronic subsystems such as computer room air conditioning (CRAC) units.

This work is focused on the energy consumption of the server, which belongs to the data center's IT infrastructure. Understanding how the energy is consumed among the components of the server is essential to define an energy prediction model. This thesis presents a system-level power model of the server, which includes the power consumption of the processor, the random-access memory (RAM), and the network interface controller (NIC).

The model is based on Lasso linear regression [67] with non-negative coefficients. The selected variables reflect the power consumption activity of each hardware component. The model is independent of the usage of the server. The test cases are created in a way to explore all the possible workloads of each hardware component. The data collection includes the regression variables as well as the power consumption associated with each measurement. We used regular approach for dividing our data set into training, testing and validation sets. The model which is derived after fitting the data with the training set, is tested for its prediction accuracy against the testing set. The validation set which is a random sample of the data collection, is used afterwards to evaluate our predictions against data which is unknown to our prediction model.

This study, demonstrates the feasibility of deriving a system level power model that can be applied for predicting the energy consumption of the server without using any available tool or utility. We show the advantage of using L1 regularization for reducing the number of coefficients in regression-based power modeling. We also provide a power model that is independent of power usage scenarios and which can be used for run time power estimation with reasonable accuracy.

1.2. Structure of the thesis

The thesis is structured into five main chapters starting with the introduction of the topic. The rest of the work is organized as follows:

Chapter 2 gives an introduction on data centers energy consumption, and briefly presents the architecture of data center. It explains the importance of energy consumption modeling and prediction, and describes how they are used to increase the energy efficiency in different areas of data center such as computing resources, network plane, virtualization layer and associated business cases. Finally, it presents power modeling and prediction approaches in processor, server and data center levels.

Chapter 3 presents the regression based power model that we use to construct the equation which predicts the energy consumption of the server. It describes the regression variables that are used during the model fitting and explains the methodology followed for deriving the model.

Chapter 4 describes how the experiment is set up. It presents the characteristics of the server which is used for data collection, and the tools that we developed and used to collect the data. The test cases which are created to cover all possible workloads are also presented in this section. Then it explains the steps which are done for fitting the model and presents the final power model. Finally, it explains the model evaluations, and illustrates the results for the accuracy of the model in different server's workloads.

Chapter 5 summarizes the work briefing the purpose of constructing the energy model for the server. It states few observations about the regression variables that are used and the results that we got. Finally, it and proposes alternative directions that could result in more accurate predictions and would test the adaptability of the model in different types of servers.

Chapter 2

Background and Related Work

2.1. Data center energy consumption

Data centers typically are powered by electricity. However, following the strategy for decreasing carbon emissions and complying with sustainable operational models, modern data centers use alternative energy sources such as geothermal, wind and solar power. The electric power flows from external power grids into internal infrastructure facilities, Information Technology (IT) equipment and other support systems. The energy flows to the internal IT facilities through Uninterrupted Power Supplies (UPS) to maintain a consistent power distribution even during possible power failures.

The architecture of a data center is complex since it does not only consist of the hardware elements but also the software that runs in the IT infrastructure. Therefore, we can categorize its elements into two layers which are hardware and software, as shown in Fig. 3. The hardware consists of many components. The major ones are cooling systems, power distribution units, lighting equipment, servers and networking equipment. The software layer can be further divided into two subcategories, the Operating System/Virtualization layer and the applications. The first mainly refer to the host OS that is installed in the servers and the cloud deployment running on top of it. The second refers to the different type of applications running in the servers which vary depending on the industry and business cases.

Understanding how the energy is shared among the elements of such a complex system as well as predicting energy consumption, requires a system optimization cycle as presented by M. Dayarathna *et al*. in [6]. Whether we want to model the consumption of the whole data center or we are particularly interested in the IT infrastructure, the general approach can be narrowed down to the following process. Initially, we need to measure the energy consumption of each component that is considered and identify where the most

energy is consumed. For that we select the features that will construct the power model. Different techniques can be used for feature selection, such as regression analysis and machine learning. The accuracy of the power model needs to be validated. Finally, the model can be used to predict the system's energy consumption, and find out means to focus on improving the energy efficiency of the data center.

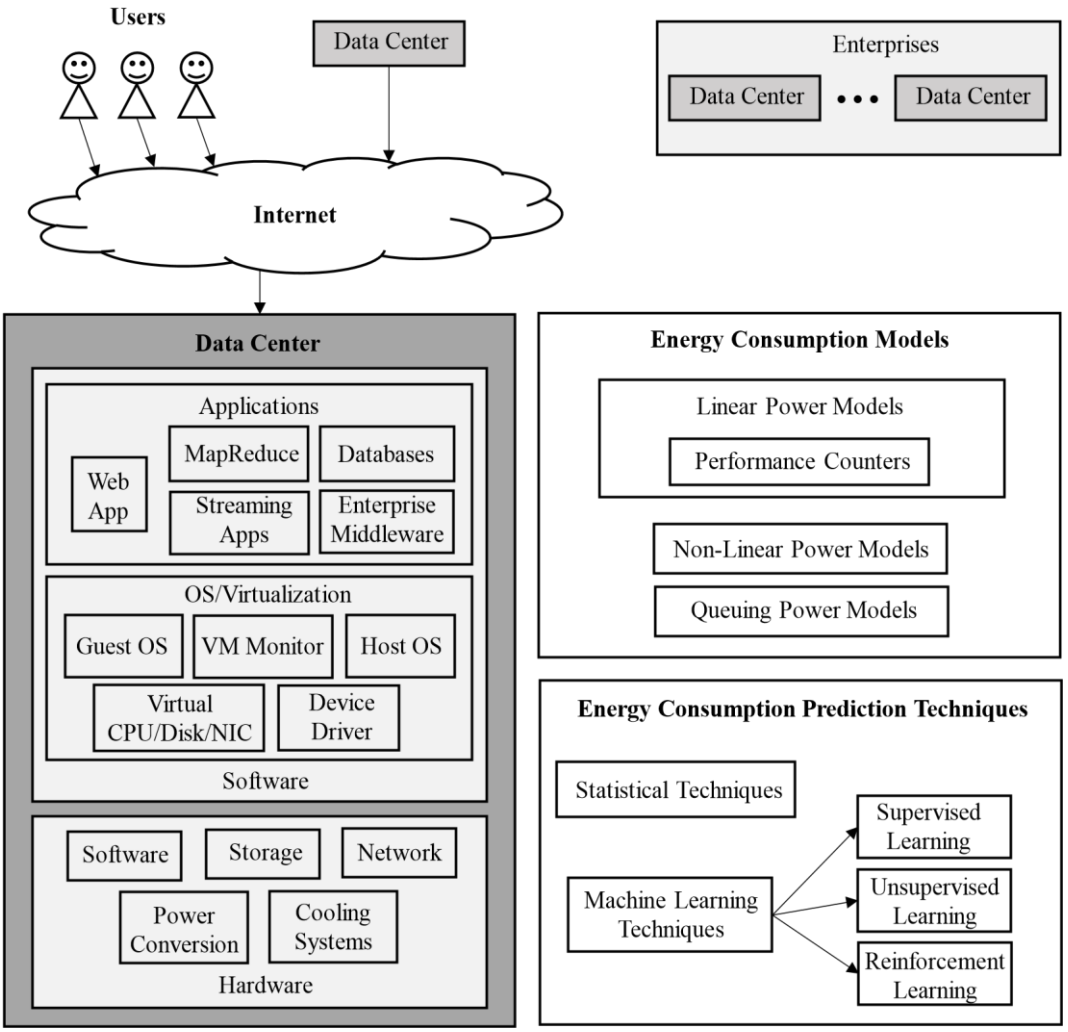

Fig. 3 A view in the context of energy consumption modeling and prediction in data centers. The components of a data center can be categorized into two main layers: software and hardware.

There has been a lot of research on energy consumption prediction for data centers Information Technology infrastructure. Initiatives and efforts to reduce the cost associated with the power distribution and cooling of the equipment are expanding, hence the power

management has become an essential issue in enterprise environments. Server takes the highest chunk of the energy consumption in the racks [6]. Consequently, there is need of understanding how the energy is consumed and what are the main components of the server which impacts the energy consumption the most. Several studies [57][59] evaluate the system level power consumption and propose models which predict the energy considering various server components. Other studies [43][50][54][55] focus on processor's power consumption. The processor is the component inside the server which consumes the most energy.

Hardware Performance Counters are proposed in many works [43][50][55] to estimate power consumption of any processor with the use of data analytics or statistical techniques. They provide significant information about the performance of the processor. The event counter mechanisms have different implementation which varies depending on the processor family, and so does the number of the available software and hardware events. Also, there are limitations on how many events can be measured simultaneously. For example, the IBM Power 3-II has 238 available performance counters, while only 8 of them can be measured simultaneously [7]. In the Intel Pentium II processor, only 2 events out of 77 can be measured concurrently [7].

2.2. Towards a green data center

There are many best practices and guidelines for achieving energy efficiency in the data centers. Nevertheless, the over-provisioning of IT resources leads to underutilized IT equipment and energy inefficiency. It is accepted that low utilization of servers, inefficient network plane management, limited virtualization adoption, and lack of business models [8] are the most significant factors that impact negative the energy usage cause by IT loads. In this context, we present solutions which try to tackle these challenges and make data centers greener.

2.2.1. Server Plane

Google stated that a typical cluster of servers is utilized on average 10% to 50% [9]. Dynamic power management (DPM) has been proposed to address the energy inefficiency

cause by underutilized servers. Dynamic voltage and frequency scaling (DVFS) is one technique for DPM which uses low voltage supply and low frequency. When there is not intensive work load in the server, neither is need to operate a processor at maximum performance [10]- [12] we can apply DFVS to save energy. Another approach is server consolidation. We consolidate jobs to a limited number of highly utilized servers, and switch the rest into low power or OFF states [13]- [17]. Job scheduling is also used to improve workload management focusing on energy conservation to achieve higher server utilization [18]- [20].

2.2.2. Network Plane

These studies [21]- [23] estimate that network infrastructure consumes 20% to 30% of the total energy consumption in a data center. Network links are highly underutilized, operating between 5% and 25% [24], and they remain idle for approximately 70% of the time [25]. K. Bilal et al. [26], explain that the conventional three-layer tree topology consumes significant amount of energy, partially because enterprise level devices consume a lot of electricity. Alternative data center topologies, such as Fat-Tree [20], Jellyfish [28] and VL2 [29], are proposed to tackle the energy inefficiency of the conventional topologies.

Virtualization techniques is another approach for better network resource utilization. Software defined networking (SDN) solutions provide dynamic resource allocation [30], and network resource scalability by adjusting the active network components of the data center [31]. Network load and energy consumption proportionality is another factor that impacts the energy consumption in the data center networks. Network devices remain underutilized; however, they consume significant amount of energy. D. Abts et al. [32], present methods that adjust the power consumption and the performance of the network based on the amount of traffic. C. Gunarante et al. [33], show that Adaptive Link Rate (ALR) in Ethernet links can maintain a lower data rate for more than 80% of the time, saving significant energy and only adding a very small inflation to the delay. The conventional tree topology or the strictly adoption of bisection topologies such as VL2, Fat-Tree and Jellyfish, are not separate solutions towards an energy efficient network plane. Data centers run different type of applications and services.

A holistic design and common approach for a specific topology does not benefit neither the quality of services nor the operation of the data center. There is need for application-specific solutions that suit the dynamics of the modern data centers and allow flexibility in energy utilization management, improving the efficiency [34]. SDN and virtualization techniques are undoubtedly game changers when reshaping and optimizing the network infrastructure. These technologies allow us to adjust the operation of the network plane based on the network load which yields remarkable amount of energy consumption. Disruptive evolution of data analytics and machine learning can provide meaningful information to the data center operators. They can predict the traffic and adjust the behavior of the network elements according to the demand, without compromising the quality of service (QoS).

2.2.3. Virtualization Plane

One of the benefits of virtualization is that we can share the available physical resources among virtual machines (VMs). Dynamic resource allocation is a technique which allow us to utilize and assign dynamically free computing resources among VMs, saving at the same time considerable amount of energy. Virtualization is an efficient way to provide server consolidation and power off the server that operate in idle mode. In many cases, an idle server consumes 70% of the power consumed by a server running at the full CPU load [35].

2.2.4. Business Model Plane

Lack of proper business models, pricing policies and conflicting priorities, are additional reason for creating energy inefficiency. Customers are not charged in proportion to their resource utilization. The lack of monetary agreements between data center owners and customers increases the interest towards an environmental chargeback model that charges tenants based on their energy consumption [36]. Microsoft has implemented such chargeback models where customers are charged according with their power usage [37].

An alternative approach that can help businesses to adopt energy practices is the collection of metrics related to energy consumption. There are various metrics which

allow us to measure infrastructure energy efficiency. The Green Grid consortium has proposed Power Usage Effectiveness (PUE). It defines the ratio of energy that is consumed for cooling and power distribution of the IT infrastructure, to the energy used for computing. PUE closer to 1.0 means nearly all the energy is consumed for computing. Often PUE does not consider all the elements of the IT infrastructure, hence it adequately reflects efficiency [38]. There has been effort to redefine the metrics that measure the energy efficiency in a better context such as power to performance effectiveness (PPE), data center productivity (DCeP), and data center infrastructure efficiency (DCiE) [38], [39].

Architecture of cloud infrastructure and multitenant virtualized environments of data centers increase the complexity of the chargeback models. Resources are shared and therefore chargeback models that incorporate flexible pricing models must be developed. There are initiatives to integrate such models with cloud infrastructure and data analytics. One example is Cloud Cruiser for Amazon Web Services [40].

2.3. Processor level power modeling

Processor is one of the largest power consumers of a server [41]. It has been shown that the server power consumption can be described by a linear relationship between the power consumption and CPU utilization [42]. There are many comprehensive power models that rely on specific details of the processor architecture and achieve high accuracy in terms of processor power consumption model. This section, describes different studies on modeling and predicting power consumption using performance counters and associated events, in processor level. These models, applied during thread scheduling and Dynamic Voltage/ Frequency Scaling (DVFS) configuration.

Rodriges *et al*. [43] estimate power consumption in real time by exploring microarchitecture-independent performance counters. They use two different CPU cores, one suitable for low power applications and another for high performance applications. Their study considers a small set of events and associated counters that have strong impact to the power consumption.