Automation of Energy Performance Evaluation of Software Applications on Servers

Jasmeet Singh, Veluppillai Mahinthan, and Kshirasagar Naik
Dept. of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, Canada

Abstract—*Although the hardware subsystems, namely, processors, memory, disk, and network interfaces of a server actually consume power, it is the software activities that drive the operations of the hardware subsystems leading to varying dynamic power cost. There are a number of ways to optimize application programs at their design stages but it is difficult for the developers to analyse their applications in terms of power cost on the real servers. In this paper, we present the design of an automated test bench to measure the power cost of an application running on a server. We show how our test bench can be used by software developers to measure and improve the energy cost of two Java file access methods. Another benefit of our test bench has been demonstrated by comparing the energy costs of compression and decompression features provided by two popular Linux packages:* 7z *and* rar. *Overall, this paper makes a contribution to reduce the perception gap between high level programs and the concept of energy efficiency.*

Keywords: energy cost, automated test bench, synchronization, energy efficient software design, file access methods

1. Introduction

Electrical energy is a key resource consumed by all computing platforms [1], [2], and the design of a software application has a significant impact on the power consumption [3]. Various techniques have been suggested to reduce the power consumption of software systems in [3] and [4]. Considering the fact that power bill accounts for a significant portion of the cost to run a data center, it is useful to analyse and minimize the energy cost of applications running on large systems, namely, servers. Although there are a number of ways to optimize the application at its design stage, developers generally do not consider the energy cost of their software while making important design decisions. They find it difficult to measure the energy cost incurred by their workload and know how it behaves on real servers inside data centers. In addition to this, the measurement process takes a lot of human effort and time.

In this paper, we present the design of an automation system, to measure the energy cost of an application running on a server, with the following properties: (i) a power automation software tool (PAST) is developed for automating the measurement process; (ii) the PAST runs on a monitoring computer which is the same machine used by the developer and different than the server under test; (iii) both the application running on a server (Load) and the power measurement instrument are remotely controlled by PAST for synchronization purpose; and (iv) for statistical data collection of power performance, the PAST can repeat a test on the Load multiple times without human intervention.

By using the automated test bench, developers can upload their application to the server and measure the energy cost of running it for the various design choices. By this way, they can concentrate more on the development, without wasting time on the measurement process. By means of our test bench, we validate the claim in the reference [5] that the energy cost of one Java file read method, FileInputStream (M1) is more than the other method, BufferedInputStream (M2). Then we study the impact of introducing a programmer *buffer* to both the methods, by measuring their energy cost of reading a file from the disk with varying buffer sizes. Although M1 consumes more energy than M2, their energy cost is same for a wide range of buffer sizes. In addition, the energy cost of M2 is further reduced after selecting the optimal *buffer size*. The other benefit of our test bench is that it can be used to compare the various functions of software in terms of their energy cost. Nowadays there are many software applications in the market providing the same functionality. The information regarding the energy cost of the same operation by different software applications allows us to chose the energy efficient ones in data centers. We analyse the energy cost of the compression and decompression features of two famous Linux packages: *7z* and *rar*.

The rest of the paper is organized as follows. In Section 2, we briefly present the related work and compare our approach with the other energy measurement tools developed recently. In Section 3, we explain the system model of the automation framework. Implementation details of the automation framework have been explained in Section 4. In Section 5, we explain how the automated system has been used to conduct experiments. Some concluding remarks and directions for future work are provided in Section 6.

2. Related Work

The techniques for understanding the power cost of servers can be categorized into three major groups: (i) *direct measurement* by means of instrumentation of the hardware [6]; (ii) *estimation* by means of power models [7], [8];

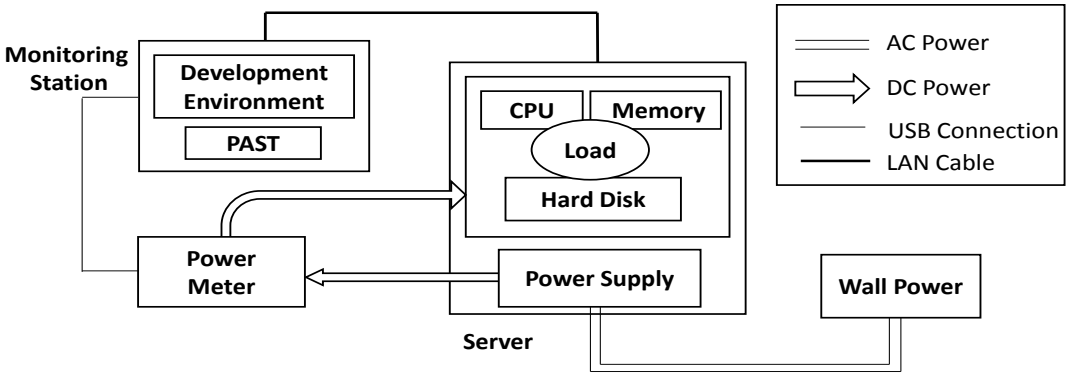

Fig. 1: System Model

and (iii) *software measurement* by means of various tools and application programming interface (APIs). A deeper understanding of power costs of computing subsystems, namely, memory, processor, hard disk, and other peripherals, enables better use of storage encryption, virtualization, and application sandboxing [9], [10]. The authors of the paper [11] studied in detail the effects of abstraction layers and application development environments on the energy efficiency of software. Their results indicate that greater use of external libraries is more harmful in terms of energy cost for large scale applications. Ardito et al. [5] developed the concept of introducing the energy efficiency into SQALE (Software Quality Assessment based on Lifecycle Expectations), one of the software quality models to monitor the impact of software on energy consumption during its development. They identified some energy efficient software guidelines and translated them into measurable requirements of the model. Although direct measurement of power consumption is expensive, it gives more accurate results than estimation models [7].

Our work also falls in the category of direct measurement. The tools used by developers to measure the energy cost of their applications rarely exist. The authors of the paper [12] presented a new tool for mapping software design to power consumption and describe how these mappings are useful for the software designers and developers. In reference [13], a comprehensive survey of different energy measurement approaches has been done. Based on this survey, the authors have come up with four recommendations for the efficient energy measurement approaches: (i) accurate measurements for better precision; (ii) fine-grained power models to trace how and where the energy is being used in software; (iii) reduce user experience impact - the measurement tools should not require manual modifications of source code of the applications; and (iv) software-centric approaches for better evolution and flexibility.

We have also reviewed those efforts similar to our framework which discuss automation of energy measurement. PowerPack [14] and pmlib software [15] have automated the energy profiling of parallel scientific workloads by software code instrumentation. These tools have a set of user level APIs which one can insert before and after the code region of interest to create its energy profile. Both these tools did not talk about the applicability of their APIs to the target code of all programming languages. PowerPack requires additional sensing resistors for each of the power lines in addition to the power meter. Moreover, these tools can not be used to measure the energy cost of closed source applications. In contrast, our framework does not need the manual modification of source code of the application for its energy measurement. In another recent work [16], they have designed a framework called software energy footprint lab, which executes the software of interest on the server and output the power consumed during the execution on a separate machine. Their approach requires manual effort to start the software under test and sending the commands to their Data Acquisition System right before the software is executed and another one right after it terminates, for synchronization. Our approach is different from them as PAST controls both the execution of the software as well as the measurement process. The process of synchronization between the server and the meter is automated in our approach. In addition, the measurement process of the same application can be repeated a number of times for statistical significance.

3. System Model of Test Bench

The system model of the automation framework has been shown in Figure 1. The definitions of all the terms used in the figure are given below.

Server: A system for which we are interested in evaluating the energy cost of running an application.

Load: A software application that runs on the server, and we measure the energy cost of running that application.

Power Meter: A data acquisition unit used for measuring power. We used a Lab-Volt 9063-00 Data Acquisition and Control Interface as a power meter.

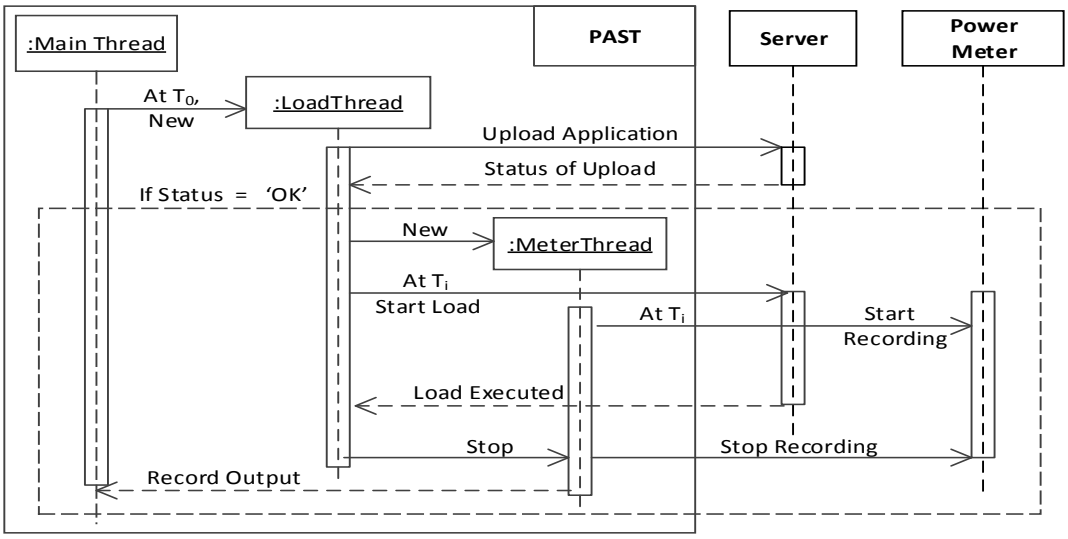

Fig. 2: Message Sequence Chart

Wall Power: Supplies AC power to the Server.
Monitoring Station: A computer equipped with the PAST system which controls both the Server and Meter. A programmer is developing his application on this machine and can run PAST to upload the application to the Server and measure its cost. By running PAST on a separate machine, it starts executing the Load on the Server as well as starts the Meter to record current and voltage values simultaneously.

The Monitoring Station is connected to the Meter via an USB (Universal Serial Bus) interface and to the Server through a LAN (Local Area Network).

Our test bench can be used to measure: (i) the power consumed by a server's individual subsystems, namely, memory, disk, and processor, if their power lines are easily accessible; and (ii) the total power cost of a server. Only the total power can be measured for a server where one cannot identify the power lines to its individual subsystems. To set up the test bench for power measurement of individual subsystems, we examined the different power lines from the ATX 24 pin connector which powers the whole motherboard of desktop computer. The power lines to the processor (CPU: central processing unit) operate at $12V$, first fed to the voltage regulator module which converts the voltage to the actual voltage required by the processor [17]. From the 24 wires of ATX connector, one yellow wire of $12V$ is feeding power to the processor. The other two yellow wires are from the ATX 4 pin $12V$ Power Connector (ATX v2.2) dedicated for the processor. The disk (Hard Disk) is getting power from a Molex 4 pin power supply connector which operates at two voltage levels, $5V$ and $12V$. The memory (RAM: random access memory) system is getting power over three lines from the 24 pin connector, and the voltage level is $3.3V$. The total power cost can be measured from the AC (Alternating Current) power lines to the server power supply.

4. Automated Test Bench

In our test bench, we use a Lab-Volt 9063-00 Data Acquisition and Control Interface system, known as the Meter in this paper. To read power samples from Meter, the device supports APIs in the form of Microsoft Dynamic Link Library (DLL). Therefore, the PAST is developed in Visual Basic. In the remainder of this section, we explain the design of PAST by means of its behaviour, which is represented as a message sequence chart, and then the key problems faced in the design of the PAST.

4.1 Message Sequence Chart

Figure 2 shows the sequence of steps of the PAST executed during the whole process of measurement.

PAST is a multi-threaded system, with three threads: MainThread, LoadThread and MeterThread. The PAST is launched on the Monitoring station with the location of the configuration file as its input parameter. A configuration file is a text file that is stored on the Monitoring Station, it contains both the Server and Meter information. Figure 3 shows some entries from configuration file. The behaviours of the three threads is described below: **MainThread:** MainThread first reads the configuration file for the $server\_ipaddress, username, password$ and the location on the server $(server\_app\_loc)$ where the developer wants to upload the application. $Launch\_app\_command$ contains the command to start an application on the server. $meter\_inputs$ in the configuration file tells which current and voltage input channels of the Meter and at what sampling frequency $(meter\_ sampling\_freq)$ the Meter should produce those values. It then starts the LoadThread and waits for the other threads to finish.
LoadThread: This first uploads the application onto the server. It stops the process if the application is not uploaded

successfully. If upload is successful, then it initializes the Meter with the meter information being read from the configuration file. If the Meter is ready to read then it starts a new thread MeterThread and starts the application on the Server.

MeterThread: This starts recording the current and voltage values from the Meter by using meter API calls. LoadThread ensures that the MeterThread is recording the values till the application is running on the server. And finally it saves all the values in to the file inside directory $(Recording\_dir)$ on the Monitoring Station.

```
server_ipaddress=192.168.1.148
username=developer
password=developer
local_application=C:\MyApp.jar
server_app_loc=/home/jasmeet/
Launch_app_command=java - jar MyApp.jar
iterations=5
component=CPU
tunable_parameter=BufferSize
tunable_parameter_array=128,256....
Recording_dir=C:\Power\Results
server=linux
meter_sampling_freq=1000
meter_inputs=E1,I1,I2,I3
```

Fig. 3: Sample Configuration file

From the recorded values, energy cost of running an application is computed by using the expression:

$$Energy\_cost = \sum_{\forall i} V(i).I(i).\Delta t \quad (1)$$

where $V(i)$ and $I(i)$ are the i^{th} voltage and current samples, respectively, and Δt is the sampling interval.

4.2 Key Challenges

There are some practical problems in measuring the energy cost of an application at the subsystem level, namely, processor, memory, and hard disk. There are only 4 current and voltage inputs to the meter. Therefore, at a time only 4 power channels can be measured. However, in case of our desktop computer, for all the three components (processor, memory and disk), there are a total of 8 power lines needed to be monitored. Therefore, we measured the power cost of the three subsystems in three repeated experiments.

5. Experiments and Results

In this section, we show how software developers can use our test bench to evaluate the energy performance of running an application on a server with various design options. We compare the energy cost of two Java file access methods: (i) M1 using FileInputStream only and (ii) M2 using BufferedInputStream. Ardito et al. [5] intuitively claim about the energy efficiency of these two methods without any measurements. First, we validate their claim by measuring the energy cost of the methods on our test bench. Then we revise the two methods by introducing a *buffer* into them and measure their energy cost with varying buffer sizes. We also compare the revised methods to read extremely large files in terms of their energy cost. Next, we compare the energy performance of two packages *7z* and *rar* with respect to compression and decompression. Table 1 shows the configurations of two machines used in our experiments.

Table 1: Server Machines Configuration

Parameter	Desktop (ASUS P4P800-VM)	Real Server (Dell PowerEdge 2950)
Processor	Intel Pentium 4, 3.2 GHz	7x Intel Xeon, 3 GHz, 4 cores per processor
Hard Disk	80 GB IDE	1.7 Tera Bytes SAS
Main Memory	2 GB DIMM	32 GB DIMM
Operating System	Linux (Ubuntu 13.10)	Linux (Ubuntu 13.10)

5.1 Example of using test bench to make important design decisions

Listing 1 and Listing 2 in Figure 4 describe M1 and M2, respectively. We measure the energy cost of CPU, memory and disk for reading a video file of size 512 MB (Mega Bytes) with M1 and M2 on a desktop machine. Figure 4 shows the results of our measurements by comparing the energy cost of all the three components for both M1 and M2. The reason behind the less energy consumption by M2, for all the components is that it reads a file of any size in larger chunks equal to the size of its internal buffer from the disk, whereas M1 reads a single byte of data in one read operation. It is clear from the results that CPU consumes the maximum energy in reading a file.

We further study the impact of introducing a programmer defined *buffer* into both the methods. Listing 3 and Listing 4 describe the modified code of the two methods, and they are denoted by M1' and M2' corresponding to M1 and M2, respectively. In both M1' and M2', line #2 shows the definition of *buffer* as an array of type byte, and its size is equal to *bufferSize*. Line #4 and #5 of M1' and M2' respectively, show that in one call *read* operation reads several bytes of data of size, *bufferSize*. Therefore *bufferSize* is a tunable parameter which the developer can vary and run these methods to read a file. We measure the energy cost of CPU, memory and Disk for both M1' and M2' with buffer size ranging from 1 Byte to 64 Mega Bytes (MB).

Figure 5 shows the evaluation of the total energy cost of all the three components for both M1' and M2'. The results in Figure 5 show that after introducing a programmer buffer into M1 and M2, the total energy cost of all the three components, is maximum at buffer size 1 byte. It started

```
FileInputStream fis = new FileInputStream(fileName);
int b,cnt = 0;
while ((b = fis.read()) != -1)
{
        if (b == '\n')
                cnt++;
}
fis.close();
```

Listing 1. **M1**: File Reading using FileInputStream

```
FileInputStream fis = new FileInputStream(fileName);
BufferedInputStream bis = new BufferedInputStream(fis);
int b,cnt = 0;
while ((b = bis.read()) != -1)
{
        if (b == '\n')
                cnt++;
}
fis.close();
```

Listing 2. **M2**: File Reading using BufferedInputStream

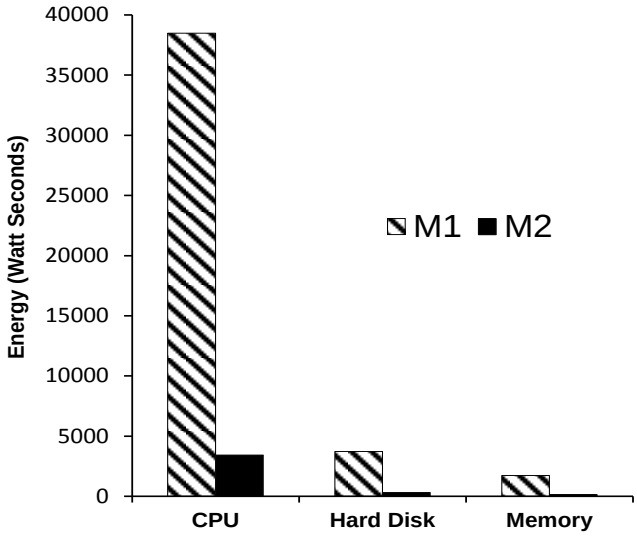

Fig. 4: Energy cost evaluation of CPU, Memory and Disk for M1 and M2 on the Desktop

```
1 FileInputStream fis = new FileInputStream(fileName);
2 byte[] buffer = new byte[bufferSize];
3 int b,cnt = 0;
4 while ((b = fis.read( buffer )) != 1)
5 {
6         if (b == '\n')
7                 cnt++;
8 }
9 fis.close();
```

Listing 3: **M1'**: Introducing user buffer in M1

```
1 FileInputStream bis = new FileInputStream(fileName);
2 byte[] buffer = new byte[bufferSize];
3 BufferedInputStream bis = new BufferedInputStream(fis);
4 int b,cnt = 0;
5 while ((b = fis.read( buffer )) != 1)
6 {
7         if (b == '\n')
8                 cnt++;
9 }
10 fis.close();
```

Listing 4: **M2'**: Introducing user buffer in M2

decreasing with the increase in the buffer size till 128 bytes. We expanded the graphs of Figure 5 in Figure 6 to show the

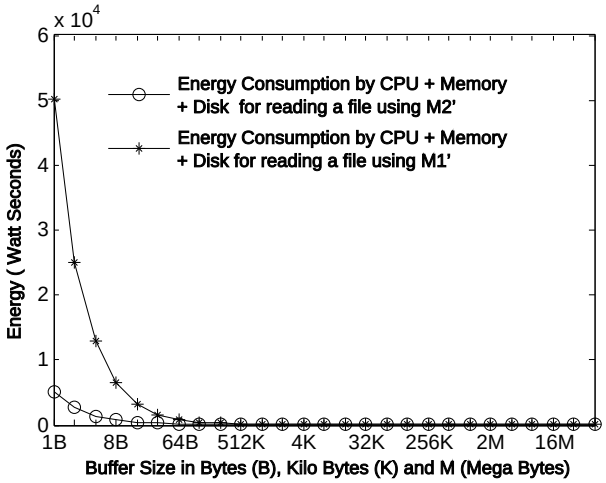

Fig. 5: Total energy cost by M1' and M2' with different buffer sizes on the Desktop

energy cost of individual components along with their total energy cost at the buffer sizes from 128 bytes to 64 MB.

Figures 6(a), 6(b) and 6(c) show the energy cost of CPU, memory and disk respectively, and their total energy cost in 6(d). The energy cost behaviour between M1' and M2' is same as between M1 and M2 for buffer sizes from 128 bytes to 8KB; in other words, energy cost of M2' remains less than M1'. Then, energy is constant for both the methods ranging from 8KB to 128KB, except that there is a sharp increase at 32KB by M1' for disk. It started increasing from 128KB to 1 MB then decreases and remains constant till 64MB. Both M1' and M2' consume almost the same energy for all the three components from 8KB to 64MB and consumes minimum energy at 16KB. Moreover, this energy is even less than M2. Therefore, it is clear from our measurements that there is a further opportunity to decrease the energy cost of M1 and M2 by introducing a programmer buffer into them. Both the methods consume almost the same energy at buffer sizes ranging from 8K to 64 MB which contradicts the claim by Ardito et. al [5] that M1 always consumes more energy than M2. In addition to this, 16K is the optimal buffer size for all the three components.

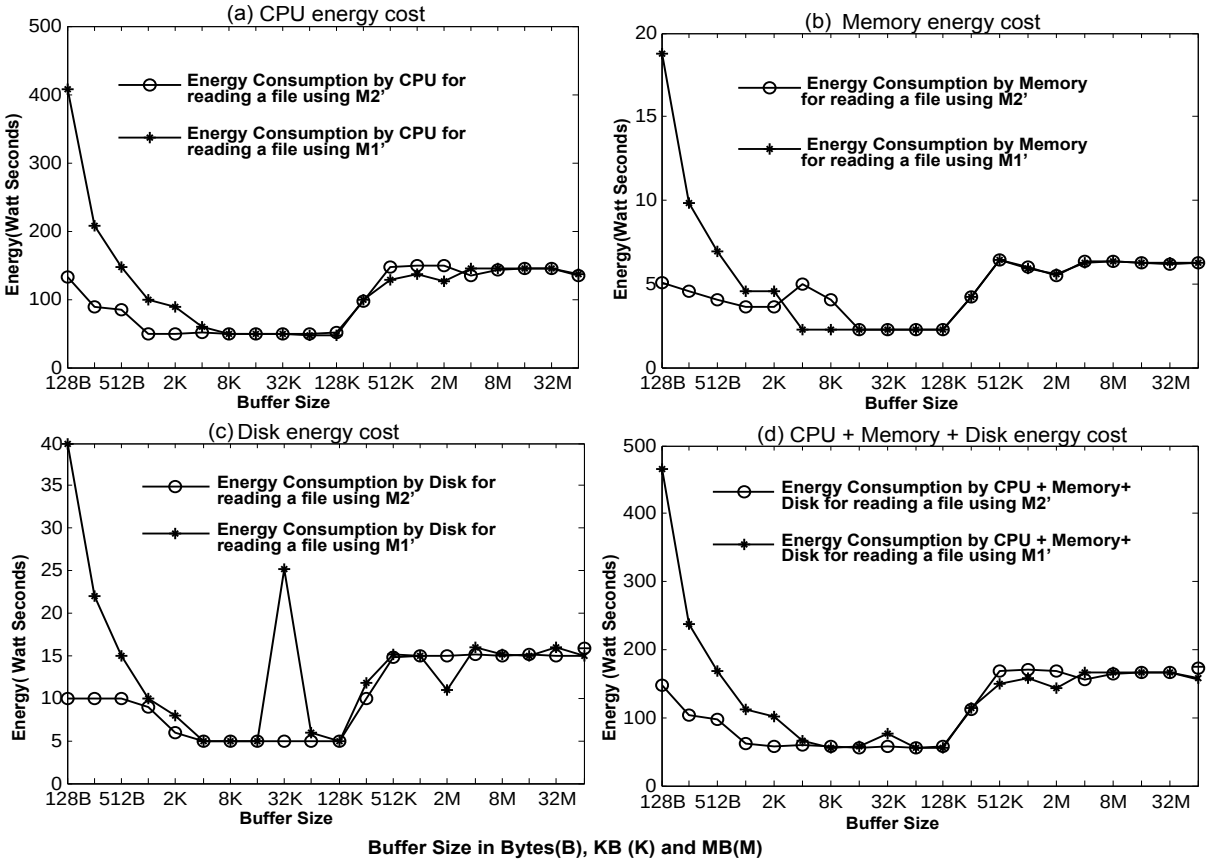

Fig. 6: Energy cost of M1' and M2' with different buffer sizes on the Desktop

To gain additional insights into the behaviours of M1' and M2' while reading extremely large files, we perform the experiments on the same desktop machine to read files ranging from 1MB to 32 Giga Bytes(GB), while keeping the buffer size fixed at 16KB. Figure 7 shows the graphs plotted for the total AC (Alternating Current) energy cost as function of different file sizes for both the methods at 16KB buffer size. It is clear from the graph that both the methods consume the same energy at 16KB buffer size. We close this section by noting the above results enable the developer to chose the right method for reading a file with appropriate buffer size during the design stage.

5.2 Using the test bench in function level energy cost measurement

As discussed in Section 1, the test bench can also be used to measure the energy cost of a specific function of an application software whether it is open source or closed source. To validate this functionality of our test bench we conducted the experiments on a real server (Table I) from a data center. We consider two popular Linux packages, namely, *7z* and *rar* to compress and decompress files. Both the packages output compressed files in *.rar* and *.7z* formats and can decompress the same to the original files. A video file of size 512 MB is used in our experiment for compression. Figure 8 shows the total AC (Alternating Current) energy cost of a server for the four functionalities of both the packages. The results show that the *7z* package consumes more energy in compressing the files to *.rar* and *.7z* formats compared to the *rar* package. However the *rar* package consumes less energy in producing *.rar* files than it consumes to produce in *.7z* format. In case of decompression from *.rar* format, *7z* consumes more energy than it consumes while converting from *.7z* format while *rar* consumes the same energy in decompressing *.rar* and *.7z* formats to produce the original files. Further investigation is required to find the causes of energy cost differences of the same operations of two packages.

6. Conclusion and Future Work

In this paper we presented an automation framework to measure the energy cost of servers while running software applications. The framework's infrastructure mainly contains a power meter, target server and control software (PAST) for synchronization and monitoring. By using the test bench, we performed actual measurements to verify the claim in a previously published paper [5] that energy cost of reading

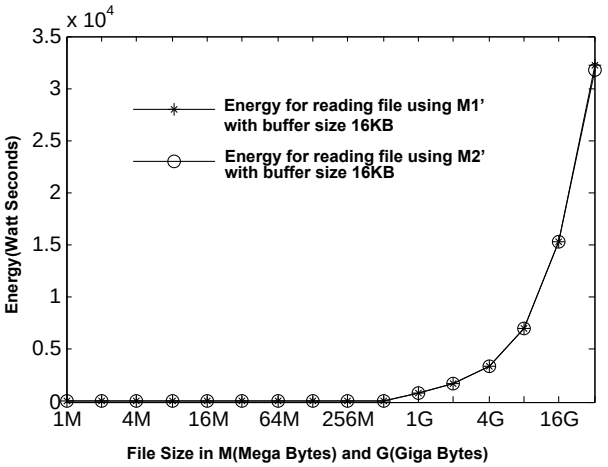

Fig. 7: Energy cost of M1' and M2' for different file sizes on the Desktop

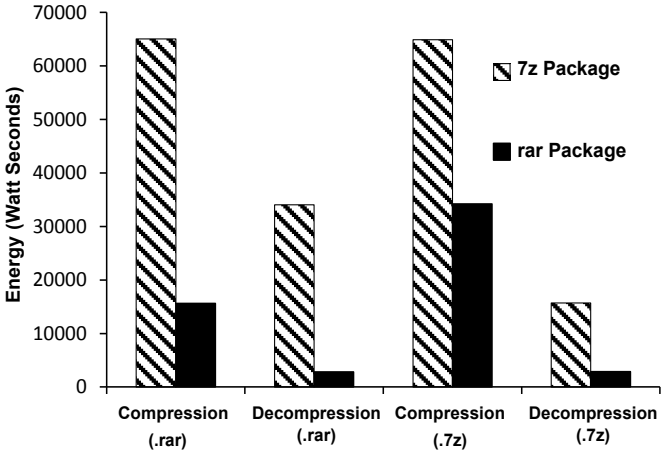

Fig. 8: Energy cost of 4 functions of *rar* and *7z* packages on the real server (Table I)

files by the method FileInputStream (M1) is greater than the BufferedInputStream (M2) method. However this claim is not valid in certain cases, if we introduce a programmer buffer in both the methods. It holds good for buffer sizes ranging from 128 bytes till 8KB, but these two methods consume almost the same energy at buffer sizes from 8KB to 64MB. Also, the introduction of *buffer* in M2 has further reduced its energy cost. Finally, we compared the energy costs of the same functionality provided by different software applications by measuring the energy costs of compression and decompression features of two Linux packages: *7z* and *rar*. The *7z* package consumes more energy than rar in compressing and decompressing files. However, *rar* consumes more energy in compressing to *.7z* format than to *.rar* format. The automation framework can be used by programmers to evaluate the energy cost of their applications. More work is required to be done to find out the causes of energy cost differences of the same operations of two packages.(Figure 8)

References

[1] T. Mudge, "Power: A first class design constraint for future architectures," in *High Perf. Computing*. Springer, 2000, pp. 215–224.

[2] K. Naik and D. S. Wei, "Software implementation strategies for power-conscious systems," *Mobile Networks and Apps*, vol. 6, no. 3, pp. 291–305, 2001.

[3] M. Sabharwal, A. Agrawal, and G. Metri, "Enabling green it through energy-aware software," *IT Professional*, pp. 19–27, 2013.

[4] D. J. Brown and C. Reams, "Toward energy-efficient computing," *Communications of the ACM*, vol. 53, no. 3, pp. 50–58, 2010.

[5] L. Ardito, G. Procaccianti, A. Vetro, and M. Morisio, "Introducing energy efficiency into sqale," in *ENERGY 2013, The Third Intl. Conf. on Smart Grids, Green Communications and IT Energy-aware Technologies*, pp. 28–33.

[6] T. Stathopoulos, D. Mclntire, and W. J. Kaiser, "The energy endoscope: Real-time detailed energy accounting for wireless sensor nodes," in *Information Processing in Sensor Networks, 2008. IPSN'08. International Conference on*. IEEE, pp. 383–394.

[7] J. C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren, and R. K. Gupta, "Evaluating the effectiveness of model-based power characterization," in *USENIX Annual Technical Conf*, 2011.

[8] Y. Sun, L. Wanner, and M. Srivastava, "Low-cost estimation of subsystem power," in *Green Computing Conference (IGCC), Intl.* IEEE, 2012, pp. 1–10.

[9] P. A. P. D. S. William, J. Kaiser, and P. L. Reiher, "Investigating energy and security trade-offs in the classroom with the atom leap testbed," 2011.

[10] D. McIntire, K. Ho, B. Yip, A. Singh, W. Wu, and W. J. Kaiser, "The low power energy aware processing (leap) embedded networked sensor system," in *Proceedings of the 5th intl. conf. on Information processing in sensor networks*. ACM, 2006, pp. 449–457.

[11] E. Capra, C. Francalanci, and S. A. Slaughter, "Is software green? application development environments and energy efficiency in open source applications," *Information and Software Technology*, vol. 54, no. 1, pp. 60–71, 2012.

[12] C. Sahin, F. Cayci, J. Clause, F. Kiamilev, L. Pollock, and K. Winbladh, "Towards power reduction through improved software design," in *Energytech, IEEE*, 2012, pp. 1–6.

[13] A. Noureddine, R. Rouvoy, and L. Seinturier, "A review of energy measurement approaches," *ACM SIGOPS O.S. Review*, vol. 47, no. 3, pp. 42–49, 2013.

[14] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron, "Powerpack: Energy profiling and analysis of high-performance systems and applications," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 21, no. 5, pp. 658–671, 2010.

[15] S. Barrachina, M. Barreda, S. Catalán, M. F. Dolz, G. Fabregat, R. Mayo, and E. S. Quintana-Ortí, "An integrated framework for power-perf. analysis of parallel scientific workloads," in *ENERGY 2013, The Third Intl. Conf. on Smart Grids, Green Communications and IT Energy-aware Technologies*, pp. 114–119.

[16] M. A. Ferreira, E. Hoekstra, B. Merkus, B. Visser, and J. Visser, "Seflab: A lab for measuring software energy footprints," in *Green and Sustainable Software(GREENS), 2nd Intl. Workshop*. IEEE, 2013, pp. 30–37.

[17] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: Methodology and empirical data," in *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2003, p. 93.