LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN

INSTITUT FÜR INFORMATIK
LEHRSTUHL FÜR SOFTWARE
AND COMPUTATIONAL SYSTEMS

Bachelor Thesis

in Computer Science

Measuring and Optimizing Energy Consumption of Verification Work on Clusters

Maximilian Hailer

Supervisor:	Prof. Dr. Dirk Beyer
Advisor:	Martin Spießl
Submission Date:	5.11.2019

Abstract

Software has become a major business branch in our society. Every enterprise concern relies on software, and so it is important to ensure the correctness of programs in usage. One way to accomplish this is the verification of software. Current tools like CPAchecker are well optimized but even on a cluster with 168x8 processing units based on an x86-architecture it can take a long time to analyze more complex programs. A side effect of this intense processing task is the high energy consumption caused by the cluster. Consequences are high maintenance costs and a significant ecological footprint. Continuing using software verification while keeping the energy consumption as low as possible would befit the serious nature of the matter under discussion.

We examine and compare multiple technologies and measurement techniques to reduce and optimize the energy consumption of clusters, especially in verification work. Data management plays an important role for the analysis and further evaluation. It comprises the steps of data gathering, saving, and retrieving. This allows checking two optimization hypotheses, one trying to decrease the base load of any cluster by dynamical scaling down the availability of computers and second, trying to reduce the overall load by a different approach, in using cluster computers based on an ARM CPU. In addition, we compare conventional three phase energy meters vs. software-based RAPL measuring in terms of accuracy and effort.

Contents

List of Figures

List of Tables

1 Introduction

Riding a bicycle, driving a car or using a computer. Everything we do needs some amount of energy. Physics shows us that there are different types of energy, e.g., mechanical, thermal, or electrical. The most versatile form is electrical energy, known as electricity. Its characteristic of being easy convertible into almost every other society-relevant energy type makes it essential for us. Not for nothing, electrification was called "the greatest engineering achievement of the 20th Century" by the National Academy of Engineering [2]. Modern industrialized countries like Germany rely very much on a stable power supply. Nearly every household has access to the national power grid. For continuously supplying the population, a supra-regional power generation strategy is required. In 2017 the energy mix of Germany consisted of 33.3% renewable, 11.7% nuclear, and 50.7% coal and natural gas or oil [3]. Consequently, more than half of the generated energy leads to climate-damaging emissions.

What does this mean for us? We should not only consider reducing energy consumption because of economic reasons, we also need to keep the ecological consequences in mind [12]. Getting to 100% renewable energy production can take a while and completely stop using electrical devices is no option for any industrial country. Therefore one solution until the energy evolution is completed would be decreasing the energy consumption.

On the other hand, we need to realize that we are living today in the 21st Century, the so-called Information Age. One major key innovation of this age among other aspects is the digitization made possible by computers and software. Using software in many different parts of our life, ranging from smartphones, cars, and even washing machines, is for the business industry a game-changer. Even in medicine, innovations like computed tomography is helping doctors to have a better look at their patients. To ensure the reliability of software, we have two leading concepts: Testing and Verification. Whereas testing covers specific scenarios, verification can guarantee that a program fulfills the given specification or not as long as the verification process is validated. So is verification better than testing? No, it depends on the specification, and as long as this is not given, a verification process is mostly not the right choice. Testing on the other side is in most cases simpler to perform and does not necessarily need specific knowledge about the system (black-box testing). For many enterprise concerns and research institutes verification work is nowadays like testing, a commonly used method to increase software quality. Tools for automatic verification like CPAchecker gets improved continuously to run faster and more efficient and also to cover more specific scenarios like testing Java programs. Running verification tools like this with more complex software, takes also a longer time to complete the verification process, which is why the Software and Computational Systems Lab (SoSy) at LMU Munich has a cluster consisting of 168x8 processing units based on an x86-architecture. Even with this amount of high-performance hardware, there are many cases where the verification task is not terminating. This can have many reasons depending on the heuristics used or still not having powerful enough hardware. Apparent solutions like expanding the cluster not only include investment costs, it also increases the energy consumption, one of the more significant downsides of having a large cluster. This contradicts the already mentioned idea of trying to decrease the energy consumption to be more climate-friendly and keep maintenance costs low. The trade-off between decreasing the energy consumption and run verification work with the same efficiency is the focus of this thesis. The way to accomplish this problem is done by optimization.

Optimization is a crucial concept used in many scientific areas. It allows getting the most out of current technologies by improving the initial idea or adding new thoughts to it. Research fields in computer science like Software Verification also make use of optimization. For this thesis, we need to take a closer look because this term has a broad spectrum of meanings. In mathematics, optimization means you minimize or maximize a cost function, depending on its arguments and additional (in)equality constraints.

$$\begin{aligned}
&\min_{x,y,z,...} f(x,y,z,...) \; or \max_{x,y,z,...} f(x,y,z,...) \\
&g_i(x) \leq 0, i = 1,\dots,m \\
&h_j(x) = 0, j = 1,\dots,p
\end{aligned} \tag{1.1}$$

Whereas generally spoken it is a methodology of making something such as a design, system, or decision as fully perfect, functional, or effective as possible (according to Merriam Webster).

There are two general ways to achieve improvement. First, by directly developing more advanced and efficient programs. Second, by generally reducing the energy consumption caused by the hardware. In this thesis, I will focus on the second point because the first one is too specific since it depends on the algorithms and data structures each verification tool uses. To realize this idea, exact measurements of the current situation are required. Measuring energy consumption needs specific hardware and software tools, which needs to be chosen best suitable for our situation. Afterward, I check the optimization hypothesis of reducing energy consumption with another CPU architecture. The goal is to achieve a relative overall decrease by building a small ARM-cluster to compare it with the Apollon Intel x86-cluster of the SoSy Lab. Before this is done, some tools for data management needs to be set up.

2 Data Management

Optimization needs a clear analysis of the current situation. Achieving this requires precise measurements over a long period. However, it is not enough to plug in any measurement device or software and start measuring. All the collected data need further processing steps to be usable in an objective, reproducible manner. It is also essential to have all the data in a compact, human-readable representation. But, what kind of data is essential to measure for our purpose? We want to know the consumption of electrical energy in the unit kilowatt-hour (kWh). The relation between the typical used SI energy unit Joule (J) and kWh is described by the following formula:

$$\begin{aligned} &1\,kWh = 1000\,Wh = 1000\,W * 3600\,s = 3.6\,MJ \\ &J(E) = E * 3.6\,MJ/kWh \text{ where E is the energy in kWh} \end{aligned} \tag{2.1}$$

Besides, we want to know the timestamp for each energy data value measured in seconds. A general way of calculating the energy difference over time is by using following formula: $\Delta E = \int_{t_1}^{t_2} P(t)dt$. Because we only measure discrete values, the formula results to: $\Delta E \approx \sum_{i=t_1}^{t_2} \Delta E_i$. This means we need to save data points of the form $(time, energy)$.

This chapter describes a general way, how to process data, generated by any measurement technique. The goal is to have the gathered and reduced data ready for later analysis or comparison. To accomplish this, we need to collect the data, extracting the relevant information, and save everything into a time-series database (TSDB)[1]. We also want to have a way to export and

[1]A time-series database (TSDB) is a software system that is optimized for storing and serving time series through associated pairs of time(s) and value(s). [9]

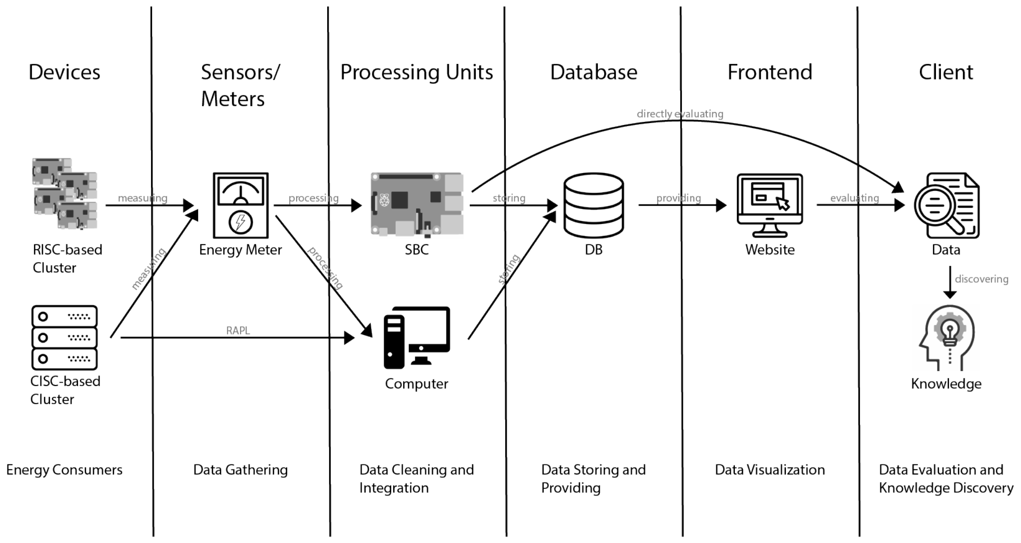


Figure 2.1: The data flow of the tracked energy consumption of any cluster, beginning from measuring, resulting in potentially new knowledge.

view the captured data. Figure 2.1 describes an overview of the processing pipeline, beginning with the raw gathered data and ending with a data visualization, which allows further evaluations and potentially knowledge discovery.

2.1 Data Gathering

At first, all available data needs to be gathered. There are two different approaches to get information. Actively asking the measurement instance having new data, this is also called polling and second, passively waiting for new data arriving. While polling has the advantage of being more controllable concerning the interval time of receiving new data, passive waiting can be more resource-friendly. So both variants are situationally applicable depending on their use-case.

2.1.1 Volkzaehler (Vzlogger)

Volkszaehler is a project of realizing your own energy counter including a software stack to access and manage the data. It consists of four modules: measurement, transfer, storing, and evaluation. In this section, we want to take a closer look at the measurement component, Vzlogger. It is compatible with many smart meters and covers among others the interpretation of the

Smart Message Language. SML is a commonly used communication protocol for most modern electricity meters in Germany [4]. Most of them have an infrared diode for read and write access to the internal data. In practice, a small computer with a USB or TTL interface like a Raspberry Pi is sufficient regarding its processing performance. It can run Vzlogger and can be connected with the smart meter via a typical IR-R/W-Plug[2] In addition it has a low energy consumption.

Vzlogger has two options for saving the gathered data: logging and putting every data point into a database. When using the log, we only have tuples consisting of time since epoch (unix time) in ms and the energy value in kWh. Extracting the latest tuple for each connected smart meter is done by a simple bash script (see A.1). For exporting all captured data, I additionally created a Python script to convert the log into a CSV file (see A.3). Vzlogger is written in C++ and runs on most embedded devices that confirm the POSIX standard. It can be run as a daemon or via cron, and it is also opensource[3]. This is why I have chosen this tool in combination with my already mentioned scripts to fulfill the task of data gathering. Vzlogger uses the concept of passive waiting, i.e. when new data arrives it automatically gets handled.

If an implementation with less overhead of the data path-through between Vzlogger and our database is required, there is also an alternative to specify some specific databases in the Vzlogger config to directly write the data. In this case, neither the bash extraction script nor other tools are required at the cost of less compatibility, because not every database software is directly supported.

2.1.2 Optional Addition: Collectd

Collectd is a daemon which collects system and application performance metrics periodically and provides mechanisms to store the values in a variety of ways. It can make the information also available over the network. Those statistics can be used to monitor systems, find performance bottlenecks, and predict future system load. One of Collectds advantages is the native compiled libraries that get loaded by one C-program. This results in better performance compared to other similar tools. Another advantage is modularity in terms of plugins. You can easily create and integrate your plugin or use a wide variety of already existing plugins. The plugin does not even need to be in C-language; it could also be programmed in Python. When combing both tools, Vzlogger and

[2]*IR-R/W-plug* means infrared read and write plug. It reduces the risk of an electric shock compared to a conventional electrical plug by only transferring light.

[3]For more detailed information, see https://github.com/volkszaehler/vzlogger (last accessed 03.06.2019)

Collectd, they result in a robust data gathering toolchain. Whereas Vzlogger retrieves the energy meter data and processes it into readable values, Collectd gathers these to provide it system-wide to other applications like InfluxDB or Graphite. For this purpose, we need to create our plugin by importing the Collectd Python library and confirm the specified protocol by implementing and registering a read function (see A.2). For more complex scenarios, we could also define a config, init, write, and shutdown function but because of the fact that we can conveniently read the latest log data via get_energy_count.sh (see A.1), we only need to call this bash script for every installed meter and add all values together to compute the overall consumption. For later availability issues, I have defined the logic that if an energy meter is offline, its value is zero. A meter is considered offline when it has no new values in the latest 10 log entries. The number 10 represents the timeout interval and can be adjusted without any problems beginning from 2 if required. Collectd uses the concept of polling. This means it is necessary to specify a read time interval. The range of this interval depends on the measuring duration and the precision quality.

2.1.3 Thinking one step further: MQTT

Message Queuing Telemetry Transport, short: MQTT can be an alternative to the toolchain mentioned in 2.1.2. It is an ISO standardized (ISO/IEC PRF 20922) publish-subscribe-based messaging protocol and often used in environments with multiple independent sensors that act as a data gatherer. The server, collecting all the data needed, is often called a broker, whereas the sensors or actors are named clients. The broker subscribes to a pre-determined list of clients that publish new data when available. Transferring this protocol to our energy meter environment would mean that we have a central instance which runs the broker software to get all the meter data. Besides, every energy meter instance needs to be connected with separate computers as long as they are not located close to each other. In addition we need to run the client program to allow publishing new information as soon as we retrieve them from Vzlogger. This setup is more suitable in a situation where we have to coordinate a bunch of distributed sensors. In the case of using one or two energy meters, this is an unnecessary additional software structure to maintain.

2.2 Data Storing

The collected data needs to be stored in some way. Otherwise, it would not be available for later usage. The most common way to do so is by setting up a database. This has the advantages of mostly redundancy free and efficient data storage and also the possibility for data aggregation. For our specific case of saving time-related energy values, we need a time-series database (TSDB). For our data processing stack, we use InfluxDB as TSDB because of its extensive programming language and SQL-like query language compatibility. Besides, it is also easy to install and has extensive application support, for example, for Graphite, Grafana, and many other IoT apps. Apart from the HTTP API, it offers support for various input plugins like Graphite, Collectd, and UDP [10]. These are the reasons why we choose InfluxDB over Graphite which is used by the Chair for Software and Computational System.

2.2.1 InfluxDB

InfluxDB is an open-source schemaless time-series database with optional closed-source components developed by InfluxData. It is written in Go programming language and it is optimized to handle time-series data. The support of SQL-like queries allows a convenient way of accessing all relevant data. For retrieving all energy values tracked in the last seven days, I created the following query:

```
SELECT mean("value")
FROM "energy_count_value"
WHERE $timeFilter
GROUP BY time($__interval)
```

The data retrieved gets later used by Grafana for visualizing purposes. With additional features like data aggregations, further analysis steps are easy to implement. Beside the excellent performance and integration ability, this makes InfluxDB the right choice for my data processing stack [5] [10].

2.2.2 Alternative: Graphite

Graphite is a open source monitoring tool. It has the capability of:

1. Store numerical time-series data
2. Render graphs of this data on demand

This monitoring tool needs to get combined with data gathering tools like Collectd to work seamlessly for our demands. In general, it is an monitoring tool that also runs well on cheap hardware. Architecturally, graphite consists of three software components:

1. carbon - a daemon that listens for time-series data
2. whisper - a simple database library for storing time-series data
3. graphite web app - A Django web app that renders graphs on-demand

The web app is uncommonly used compared to its alternatives like Grafana because of its lower customisation capabilities in terms of dashboard design. Its data storing module whisper is a fixed-size database, similar in design and purpose to RRD (round-robin-database). It provides fast, reliable storage of numeric data over time. Whisper allows for higher resolution (seconds per point) of recent data to degrade into lower resolutions for long-term retention of historical data.[4] Because this DB library is written in Python it is 2-5 times slower than RRDtool in terms of fetch operations and up to 9 times slower than InfluxDB in terms of query performance. Also, InfluxDB has compared to Graphite a significant better disk compression, which is essential for long measurement intervals where lots of data get captured.[5] Graphite includes almost everything you need; however, compared with other similar tools its components are less practical for energy measuring.

[4]According to: https://graphite.readthedocs.io/en/latest/whisper.html (last accessed: 21.10.2019)

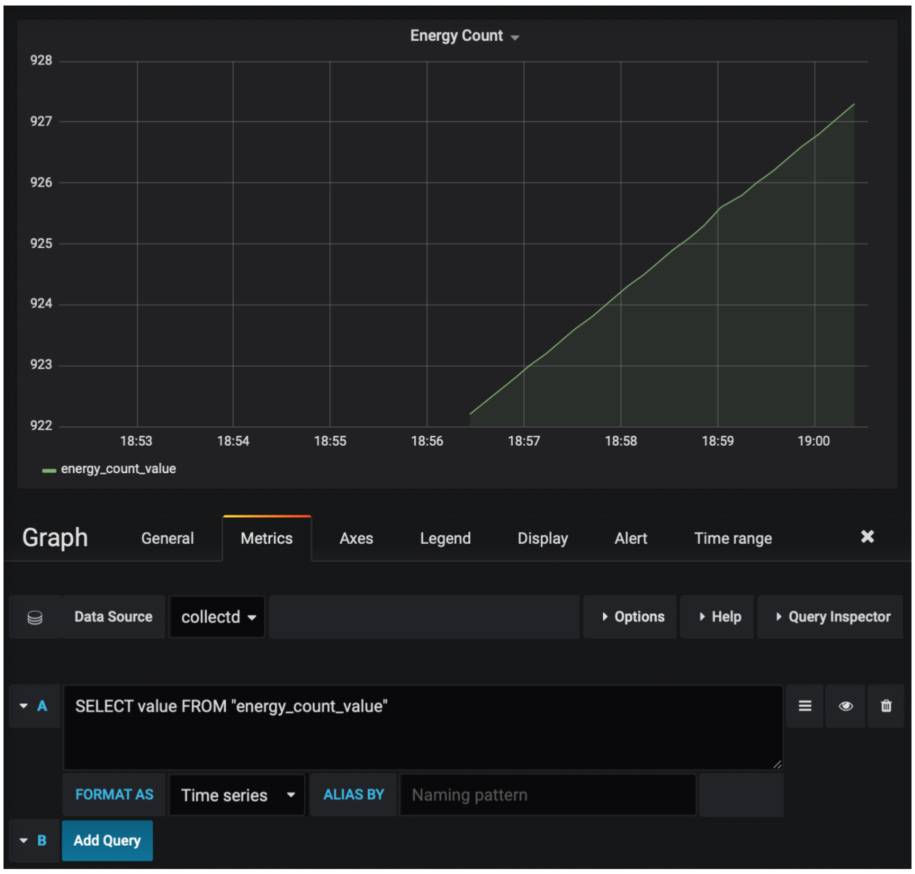
[5]Claimed at: https://www.influxdata.com/blog/influxdb-outperforms-graphite-in-time-series-data-metrics-benchmark/ (last accessed: 21.10.2019), see also https://github.com/influxdata/influxdb-comparisons (last accessed: 21.10.2019) for the test methodology

2.3 Data Visualization

Finally, our stored data needs to be visualized to allow evaluation by the user. For small measurement this step is not ultimately required, since the amount of stored data is manageable by the human eye, when just printed out in a textual form. For more extended time periods, we need a proper data representations like graphs or histograms. Nevertheless, to keep an overview, we additionally want to have data aggregation methods and the possibility to freely define the boundaries of the depicted time interval. All of this should be done in a web-interface for conveniently accessing the data from any state-of-the-art web browser. To accomplish this, a toolkit like Grafana is used.

2.3.1 Grafana

Grafana is a software library for time-series analytics. It allows the user to query, visualize, alert on, and understand the given metrics. This visualization tool runs on every relevant operating system, i.e. Windows, Linux, macOS and is compatible with InfluxDB, Graphite, and many other data source tools. Features like user authentication and organization allow an easy way to administrate a whole enterprise. The customization ability of the dashboards in combination with multi dashboard support and import is excellent for maintaining multiple diverse data sources. In addition, it has a plugin support to add more specific features. So it is possible to install extra panels like a bubble chart or heat maps. Setting up Grafana is very easy and user-friendly. When accessed with the web browser, we only need to add our InfluxDB data source and a proper dashboard for showing the data. We specify in the panel settings our query and the viewing options. When everything is set up, it is possible to track the energy consumption and power usage at any time and depict it as a comprehensive graph. For a straightforward testing scenario, of measuring a standby screen with a smart energy meter, which already has a cumulative energy count offset of 922 Wh, we see in figure 2.2 a graph with an almost constant slope, whereas the slope describes the power usage. This graphical interpretation of the data helps to evaluate and compare the upcoming measuring methods and optimization hypothesis.



Data Source collectd

```
SELECT value FROM "energy_count_value"
```

FORMAT AS Time series

Figure 2.2: Cumulative energy consumption graph with an offset of 922 Wh generated by Grafana while measuring a standby screen with an energy smart meter.

2.3.2 Alternative: Volkszaehler

As we are already using Vzlogger as a data-gathering tool, we also could use the integrated web-frontend module from Volkszaehler. It relies on its middleware tool for storing all the captured data. The whole module constellation is depicted in figure 2.3. When installed, we first need to add channels to utilize the visualization tool. This is done by specifying the value type, resolution, depiction style, and UUID. The last one is an identifier for specific channels to import them into the frontend. There are additional export possibilities like CSV or JSON. All in all the Volkszaehler module stack is a valid alternative to managing the energy consumption with one smart meter. However, compared with my suggested tool stack of Vzlogger, optionally Collectd, InfluxDB, and Grafana there are limitations in functionality. First, the Volkszaehler module

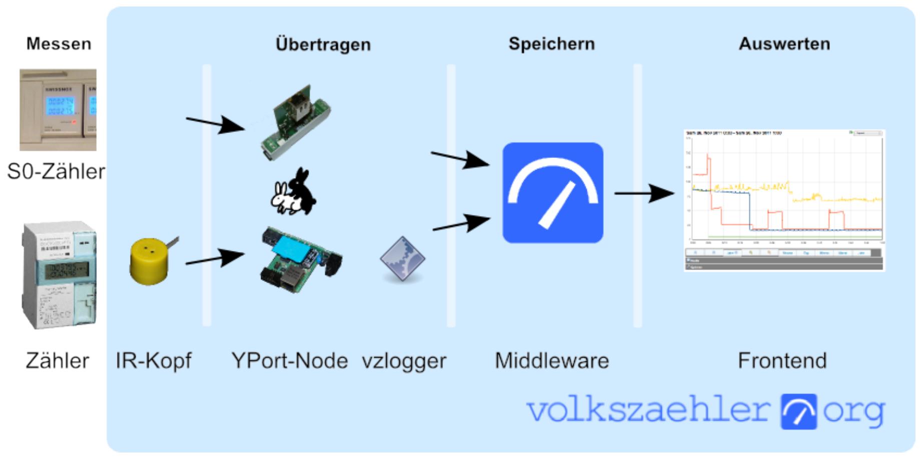


Figure 2.3: Volkszaehler diagram consisting of four modules Measuring, Transfering, Storing and Evaluation. Source: https://wiki.volkszaehler.org (Last accessed: 03.11.2019)

stack depends on smart energy meters, consequently when using software-based solutions like RAPL in exchange for the data gathering tool Vzlogger, it does not work, because we can see in figure 2.3 the front end relies on the middleware, and this relies on Vzlogger. Second, there are compatibility issues in terms of using multiple energy counters, though it is possible to show both graphs individually, we cannot add both together or use other specific aggregation methods.

3 Measurement Techniques

When our data management infrastructure is ready, it is time to begin the measurements. At first, we need to discuss which kind of techniques best fit our purpose. Comparing different techniques requires defining specific criteria [1]. For our case, there are two main points to take into consideration: accuracy and effort. The qualitative performance characteristics of the measure called accuracy consist of trueness and precision as depicted in figure 3.1. The trueness represents the degree of proximity between the measurement of a quantity and that quantity's actual true value [7]. The effort describes the cost of the measurement instruments, the maintenance costs and the total human work to install and perform the measurements. In general, a higher accuracy results into more measuring effort, on the other hand, we want meaningful results. Achieving this means to keep the uncertainty of any measurement, which is derived from the error, as low as possible. The error can be classified into human error or technical error. The first type of error depends on the experimenter. This error kind can be eliminated by multiple measurement repetitions and techniques choices. However, this is not sufficient, and the measurements should be reproducible with the same instruments by any other human experimenters to safely exclude this kind of error. Technical errors can be broken down into systematic errors and random errors. Whereas systematic errors occur on wrong instrument calibration, random errors depend on the measurement instance itself. Therefore another characteristic is vital to take into account: precision. The precision also called repeatability is the degree to which repeated measurements under unchanged conditions show the same results [7]. To sum it up, the optimal case is high precision and trueness which leads to high accuracy combined with low effort.

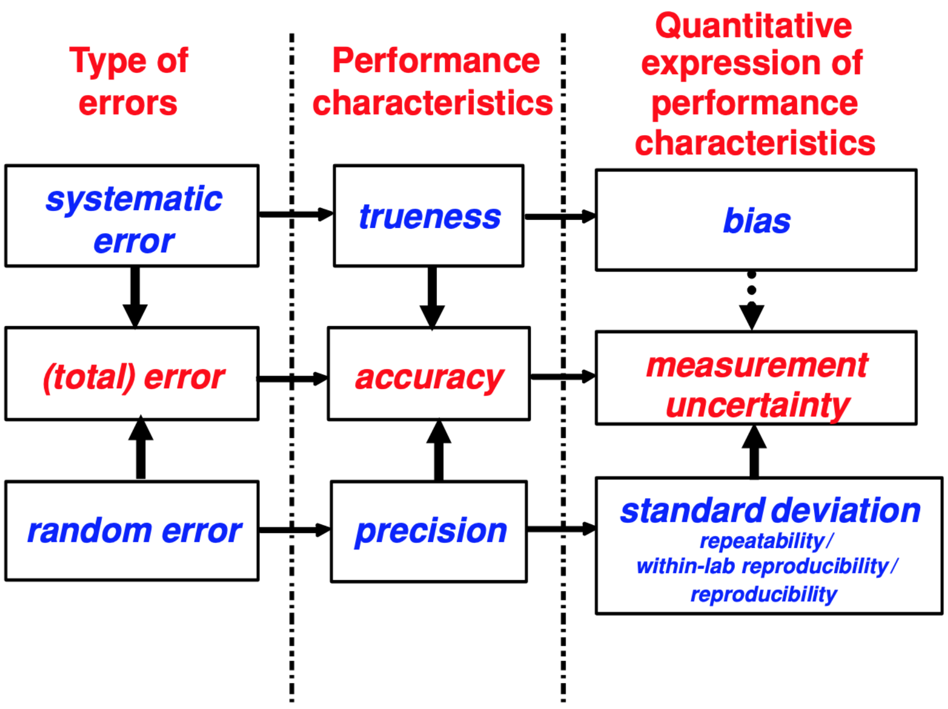


Figure 3.1: Relations between type of error, qualitative performance characteristics and their quantitative expression [7].

In this chapter, we take a closer look at two different methods of measuring energy consumption. A common solution, using an energy meter assisted by a Single Board Computer (SBC) for achieving high accuracy values and a software-based using hardware features, less effort approach called RAPL. In addition I want to evaluate an indirect method of measuring energy consumption.

3.1 Energy Meter with Raspberry Pi

Conventional electricity measuring uses dedicated devices, so called meters. They get installed before the target system and deliver information typically via an integrated optical display or using a specific interface and encoding. Because of the reason that we want further data processing steps for advanced data analytics, as already discussed in chapter 2, we need a so-called smart energy meter. We use two household electricity meters (HEM) named ED300L manufactured by EMH metering [8]. The measurement accuracy for HEMs is defined by the EN 50470-3 class index, ranging from A to C. In our case, the ED300L is specified to comply with class A corresponding to an error of ± 2.0% when operated within its rated electric current and temperature range [13] [14]. According to the user manual, measurements with a current of 60 A by 230 V voltage (corresponds to the power usage of 13.8 kW) is possible. For measuring a big cluster of computers and also for conveniently replacing one of these meters without interrupting the current flow, a second energy meter is required. Additionally, the ED300L has an infrared read-write connector for transferring the data via SML to any compatible device. This optical data interface has a data package transfer rate of 1/4 Hz to 1 Hz with a resolution of 0.1 Wh [8]. The variation is caused by the workload of the internal processing chip of the energy meter. Since we measure a computer cluster in a more extended period, at least for some hours, neither the transfer rate nor the resolution puts any significant restriction on our measurement. In detail calculation:

We expect values in Wh range, but at least 10 Wh, so the fluctuation of ± 0.1 Wh results in a relative uncertainty of 0.1 Wh / 10.0 Wh $\leq$ 1.0%. Also, we measure in almost every scenario for at least 15 minutes, so maximum fluctuations of ± 4 s results relatively to an uncertainty of 4 s / (15 * 60 s) $\leq$ 0.5%. Compared to the measuring error of the device itself of 2.0% the additional maximum uncertainty of 1.5% adds up to:

$$m(x) = x \pm \sqrt{0.020^2 + 0.015^2} = x \pm 0.025x \tag{3.1}$$

When x is the measured value this means we have a max error margin of 2.5% for most of the following tests. The real-world value x_{real} should be within the measurement interval m(x).

For the purpose of data processing, we need a computer. In our case, a so-called Single Board Computer (SBC) is sufficient. It has the advantages of being small and energy-efficient. Moreover, a SBC is significant cheaper compared to a traditional computer. I decided to use a Raspberry Pi 3B+ as SBC because of multiple reasons. First, it has all relevant interfaces: WLAN, Ethernet, USB, Serial. Second, it supports all relevant programming languages like Python, C, and command languages like bash. Also, it has enough processing power for the tasks described in section 2. In addition it has a good availability on the market and is supported by big companies. When all software components which build up our data processing stack are installed like in section 2 explained, the Raspberry Pi needs to get connected to the energy meters. To realize the connection between both devices, an IR-R/W-Plug is required.

In my case, I have used two TTL-IR-R/W-Plugs[1], one connected to the serial port of the Raspberry Pi, the other connected to the USB port with an FTDI FT232RL TTL to USB Adapter since the Raspberry Pi only has one hardware-accelerated UART pin combination consisting of TXD and RXD.

[1]More detailed specifications of the IR-R/W-Plug are described in the volkszaehler wiki: https://wiki.volkszaehler.org/hardware/controllers/ir-schreib-lesekopf

3.2 RAPL

Intel introduced the Running Average Power Limit (RAPL) feature with the Sandy Bridge microarchitecture. It is available since 2011 for all new Intel Core CPUs (2nd generation or newer) and Xeon server-level CPUs. RAPL provides sensors that can be configured and examined by reading Model-Specific Registers (MSRs) [6]. These sensors measure the energy consumption of the whole CPU package consisting of cores, uncores (all modules within the CPU package except the cores) and DRAM, as you can see in 3.2. The fact of only measuring the CPU package and DRAM raises the question: Is the energy consumption of the remaining computer modules (mainboard, any IO device, PSU) while on idle or verification work constant over a long period of time? Fulfilling this requirement is the prerequisite of using RAPL for our purpose since it is not possible for us to measure the energy consumption exclusively of any of the mentioned devices. Knowing the constant power offset leads to an easy transformation of the measured values by adding this offset to the gathered data to gain the real energy consumption of the system. For obtaining these values, there exist different solutions for reading the current energy consumption via RAPL. There is a multi-platform graphical interface solution like the Intel Power Gadget. It supports sample rates of up to 1 ms and additionally tracks the frequency, temperature, and utilization. With its additional export to CSV feature, it is possible to make useful measurements. However, its lack of automation and integration raises the need for a modifiable code-based solution. Therefore we use a C program designed by Intel to track energy consumption. To integrate this program to my data stack discussed in 2, I had additionally designed a Collectd plugin. All in all, it is possible to evaluate the measurements in a similar way as in section 3.1. This allows an appropriate comparison of both mentioned measurement techniques.

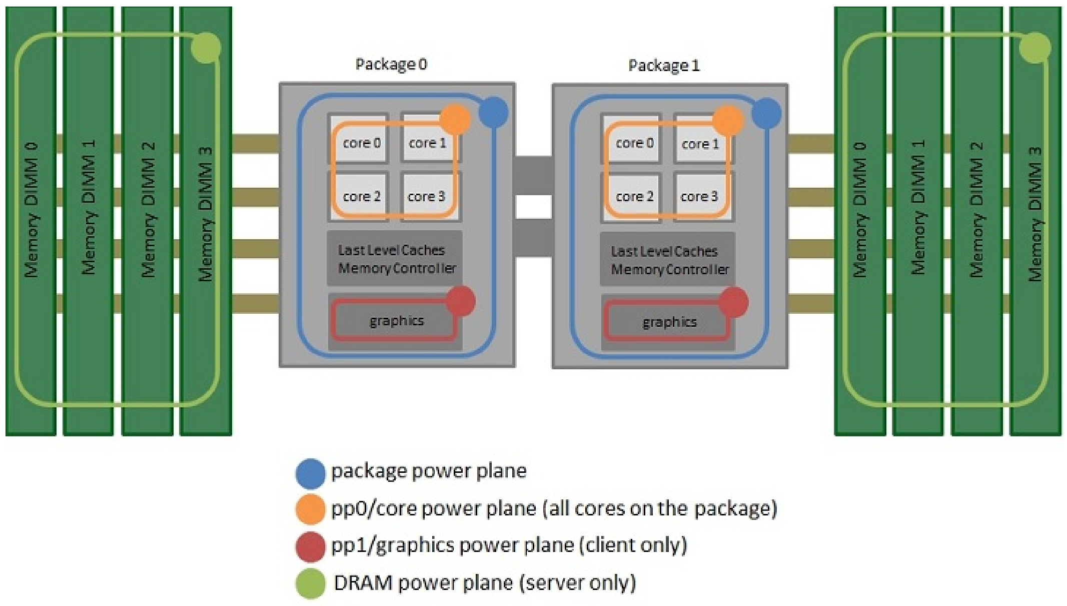


Figure 3.2: Example scheme of a multi CPU which shows the readable RAPL domains. Source: https://software.intel.com/en-us/articles/intel-power-governor (Last accessed: 21.10.2019)

3.3 PSU

An indirect method of measuring energy consumption is by using appropriate Power Supply Units which supports a digital interface of accessing the voltage and current values used to supply the system with power. Since the power is defined by $P(t) = U(t) * I(t)$ and the energy can be calculated by $E = \int P(t)dt$, we can quickly determine the energy consumption. The PSUs used by the Apollon cluster allows us to read the Ampere values for the output current at 12 V voltage.

3.4 Evaluation

To compare all three measuring techniques we need some appropriate tests. Since we are interested in verification work, I decided to utilize CPAchecker 1.8 as a verification tool combined with Benchexec to create reliable benchmarks with the help of containers to isolate the processes. In addition, I have chosen the VerifierCloud 0.700-207 to schedule all tasks evenly on all cluster nodes. The Apollon cluster serves as verification hardware. It consists of 168 Intel Xeon 1230 v5 with 33 GB RAM. We want to test the idle, full load, and mixed power usage for at least 8 hours.

3.4.1 Idle Test

In these tests, we see different power usage values depending on the testing method. This is caused by the way each measuring technique works. The Energy Meter measures the whole systems energy consumption which includes the PSU, mainboard, processor, RAM, SSD, etc. whereas RAPL measures the processor and RAM. The PSU measuring is done by the following formula:

$$\text{Power } P = \text{ Output Voltage U} * \text{Output Current } I \tag{3.2}$$

Both values I and U are measured by the PSU. As we see in table 3.1, the PSU value is much lower than the Energy Meter one because it is just the outgoing current with 12 V voltage. In addition, there are some power losses in the voltage transformation.

Idle Power Usage			
Testing Method	Duration	Avg. Power Usage	Difference to EM
EM	8 h 0 m 0 s	2.86 kW	0 W
Intel RAPL	8 h 0 m 0 s	445 W	2.42 kW
PSU	8 h 0 m 0 s	1.86 kW	1.00 kW

Table 3.1: Comparison of the measurement techniques Energy Meter (EM), Intel RAPL and Power Supply Measurement (PSU) of the idle power usage of the Apollon cluster.

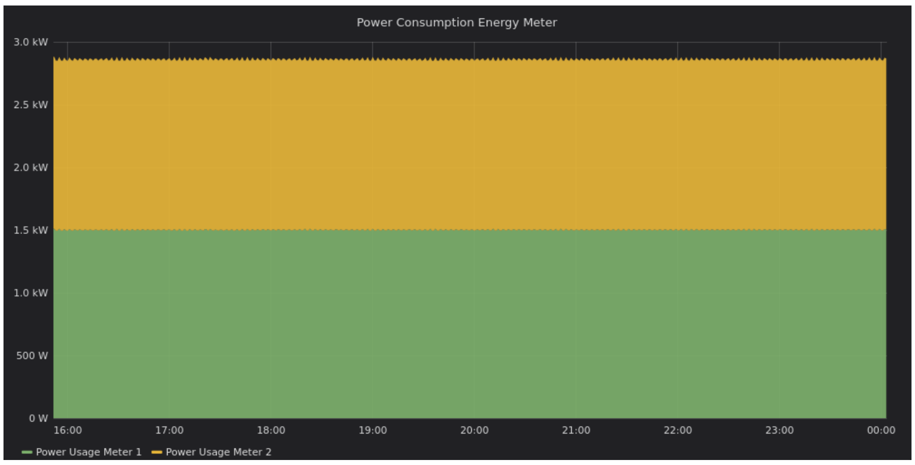


Figure 3.3: Power usage graph captured with the energy meter which shows the almost constant propagation within a specific time range of about 8h.

3.4.2 Constant Workload Test

For the constant load test, I have chosen multiple random svcomp19 tests running for more than 10 hours. To assure the constant load on all nodes of the cluster I measured the power usage within a fixed time span of 8 hours, where all nodes had almost always work to complete. The Energy Meter power usage went up significantly to 5.82 kW (see table 3.2). This is almost twice the value compared to the idle test. Apart from this, the RAPL measuring has an even higher relative increase of almost 700%. This is a clear indicator of a intensive computation task. If we take the PSU Difference to Energy Meter value into consideration we can see it stays almost constant.

Constant Load Power Usage			
Testing Method	Duration	Avg. Power Usage	Difference to EM
EM	8 h 0 m 0 s	5.82 kW	0 W
Intel RAPL	8 h 0 m 0 s	3.03 kW	2.79 kW
PSU	8 h 0 m 0 s	4.75 kW	1.06 kW

Table 3.2: Comparison of the measurement techniques Energy Meter (EM), Intel RAPL and Power Supply Measurement (PSU) of the constant random svcomp19 workload power usage of the Apollon cluster.

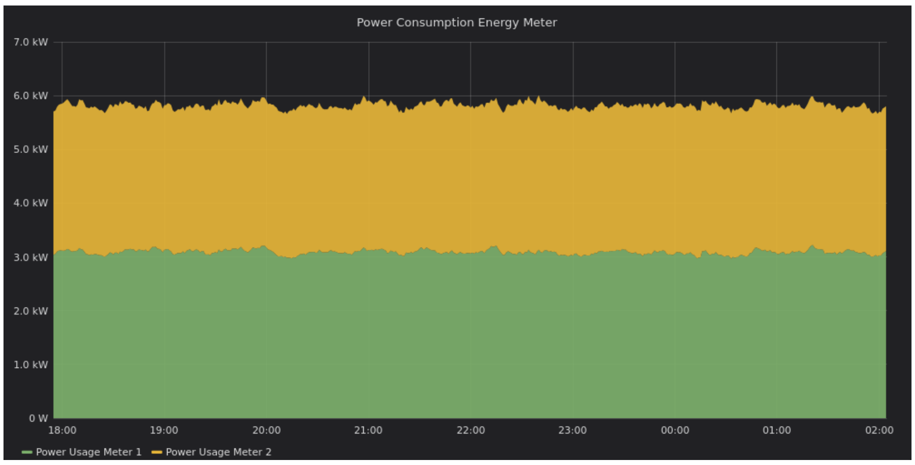


Figure 3.4: Power usage graph captured with the energy meter which shows the almost constant propagation within a specific time range of about 8 h.

3.4.3 Variable Workload Test

This test both mixes some aspects of the constant load test, an idle test. Again I have chosen to run some random svcomp19 tests but one test set after another, always with a break with the random duration within each run collection. Since the Difference to EM data points vary considerably because of the variable load, I additionally calculated the standard deviation to determine the fluctuation, as you can see in table 3.3. Less relative standard deviation means that the respective measuring techniques behave more like the Energy Meter.

Very noticeable is the relatively low deviation of the Difference to EM measuring values for the PSU method.

$$\begin{aligned} \text{Mean value } &= (1001\,W + 1064\,W + 942\,W)/3 = 1002\,W \\ \text{Standard deviation } &= (1\,W + 62\,W + 58\,W)/2 = 61\,W \end{aligned} \tag{3.3}$$

Considering the previous tests we can assume an approximate difference to EM of about 1.0kW. The difference to EM value for the RAPL measurement rises whenever the average power usage of the EM increases. This means that there is no constant offset for the RAPL measurement to extrapolate the whole power usage of the system. Consequently, there are more system components except for the CPU and RAM which has a higher power usage when there is a workload. One distinct component could be the CPU fan. Other system parts like the mainboard and SSD are also possible to cause this issue. Nevertheless, we could gather more data points and try to extrapolate with a higher grade polynomial function, but the usage of this result would be system and software specific.

Variable Load Power Usage				
Testing Method	Duration	Avg. Power Usage	Difference to EM	Standard Deviation
EM	14 h 0 m 0 s	3.81 kW	0 W	0 W
Intel RAPL	14 h 0 m 0 s	1.27 kW	2.53 kW	355 W (14%)
PSU	14 h 0 m 0 s	2.86 kW	942 W	176 W (19%)

Table 3.3: Comparison of the measurement techniques Energy Meter (EM), Intel RAPL and Power Supply Measurement (PSU) of the variable random svcomp19 workload and idle power usage of the Apollon cluster.

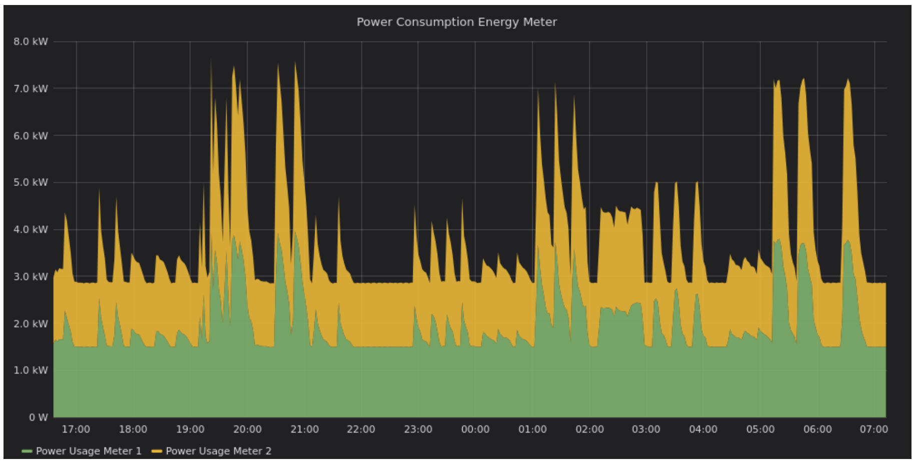


Figure 3.5: Power usage graph captured with the energy meter which shows the variable propagation within a specific time range of about 14 h.

3.5 Comparison

Both RAPL and the PSU measuring methods have their advantages but also disadvantages compared to a conventional Energy Meter measuring technique. RAPL, as well as the PSU method, does not need any additional hardware to run. Downsides are unknown or wide accuracy ranges. According to [6] the RAPL RAM measurement matches between a 20% range. Both methods are system-specific, so it is difficult to compare for example, an ARM SBC with an Intel CPU server cluster. For determining the real system consumption the PSU method could be used with extrapolation by merely adding a specific constant, nevertheless, it is not as accurate as of the conventional method. This is why we only use the energy meter for the following optimization tests.

4 Optimization

There are several ways of optimizing the energy consumption of clusters. In this chapter, I want to describe a software-based solution by dynamically shutting down unused computers. And on the other hand, I want to evaluate a hardware-based solution by using an alternative CPU architecture as computer nodes in the cluster.

4.1 Dynamic Scaling

Having a big cluster of high-performance computers also has its downsides. When there is no need for processing any relevant tasks, all cluster nodes are typically on an idle mode. This means that the computers are immediately ready to run new tasks but on the other hand, this state requires a significantly higher energy consumption as if they would be turned off. For example, the Apollon cluster at SoSy-Lab has a power usage of 3kW when all nodes are on idle (see measurements in section 4.3.6). Compared to the peak load of 8kW, the idle mode has a portion of 37.5%.

A solution to this problem is shutting down most of the unused computers and turning them on when they are needed. This idea is called dynamic scaling and is also a widespread solution for cloud computing. To realize this approach, we need a master host which is capable of turning all relevant cluster nodes on and off [11]. Shutting down can be done by a simple shutdown command via ssh. An easy way to boot the computers is by using Wake on LAN (WoL). Therefore the only technical requirement, besides WOL support, is network reachability over LAN between the master host and its clients so that the

WoL magic packets and ssh commands can be reliably delivered. Additionally, scheduling software is needed to decide which cluster node should be available. There are some policies the scheduler needs to be aware of:

1. A cluster node which already runs some tasks is never allowed to be turned off.
2. If a cluster node is long enough in idle mode, it should be turned off.
3. If any job is scalable on all potentially available computers, every cluster node should be turned on.
4. A specific amount of cluster nodes should always be available.

Besides these strict rules, there should be some prediction methods depending on the tasks and time. Usually, at night, there is no high demand for computing power, so that most computers can be shut down. When running a svcomp test-set, most computers are needed to process it quicker.

In general, having a dynamic scaling system has almost only advantages if the scheduler is appropriately adjusted on our requirements. Even in the worst case, no computer is turned off by the scheduler, there is still no disadvantage compared to if maintaining the system without a dynamic scheduling approach. The only real drawback is some more waiting time when needing a lot of compute power, since the cluster nodes have to be turned on first. Because every computer can boot up at once this waiting time does not depend on the number of machines needed and is constant. Therefore we can safely say that when using a dynamic scaling system we have the potential of reducing the energy consumption of any cluster, which has some idle time.

4.2 ARM-based Cluster

ARM processors are often used in the mobile smartphone market. One significant advantage of these CPUs is the power efficiency for usual smartphone tasks like taking photos, viewing and processing videos and long standby phases. So it is reasonable to claim the following hypothesis: ARM CPUs have the potential to be more power-efficient in verification tasks than conventional x86 server CPUs. To verify this statement in a practical way, we use the following test setup: An ARM cluster from PINE64 consisting of 7 SOPINE A64 compute modules mounted on the SOPINE Clusterboard versus the Apollon cluster which has 168 Apollon computers equipped with an Intel Xeon E3-1230 v5 clocked at 3.40 GHz with 33GB RAM. The SOPINE modules use a Cortex A53 Allwinner A64 Quad Core SOC CPU clocked at 1.1 GHz with 2 GB LPDDR3 RAM.

The verification framework we use for our testing purposes is CPAchecker version 1.8 combined with a modified version of the VerifierCloud 0.700-207. The necessary modification is described in A.4. Some libraries CPAchecker uses are only available on x86 platforms, for example, MathSAT5. This splits the testing into two more categories. One testing category for determining the power efficiency depending only on the hardware difference, and one is testing the same but with different software libraries to have an overview in real-world scenarios. Both clusters are connected with my previous discussed measuring system: Energy Meter + Raspberry Pi including my software tool stack: Vzlogger + Influxdb + Grafana.

As verification test scenario, we use following svcomp19 test categories[1]: overflows, memsafety, termination and reachsafety. Running the whole svcomp19-reachsafety test would take too long for the Pine cluster, wherefore I have reduced it to the following subcategories: ArraysReach, ControlFlow, HeapReach, Recursive, and Loops. We want to compare the verification work per power consumption ratio. Since it is difficult to compare different verification work tasks and describe it in a numerical way we keep this variable constant for both CPU architectures, so the significant value is the power consumption for the same amount of verification work.

Since it is clear that the Intel cluster will be much faster than the ARM-based one, it is also important to determine the scaling factor of the SOPine modules. So we know if the Pine system has the potential when it has enough nodes to be as fast as the Apollon cluster, to exclude the disadvantage regarding the cluster throughput. In practise it has a great significance not only to be power efficient, it is also good to be time efficient. Therefore the scaling

[1]Source of the tests: https://github.com/sosy-lab/sv-benchmarks ea7880b

factor is defined by the following formula:

$$\begin{aligned}&time(i) = \text{total time needed to complete all the tasks with i cluster nodes}\\ &scaling(i, j) = (time(i) * i)/(time(j) * j), \text{ where } i \leq j\end{aligned} \tag{4.1}$$

The scaling is normal when following statement is always true:

$$\forall i \in \mathbb{N} : \exists j \in \mathbb{N} : scaling(i, i * j) = 1 \tag{4.2}$$

Calculating the scaling factor for each possible value is practically impossible, why we measure the duration of the svcomp19–overflows test with every possible node amount setup, ranging from 1 node to 7 nodes. If the values are close to the numerical value 1, then we know that for small amounts, it is efficient to add more SOPines to boost the verification process via the VerifierCloud software.

In addition, we want to test the idle consumption of both clusters to determine the minimal cost of maintaining both computer arrays.

4.3 Evaluation

4.3.1 Pine-Efficiency Scaling Test

The scaling tests purpose is to give an indicator for normal scaling behavior of the Pine cluster when using the VerifierCloud as workload scheduler and manager. Table 4.1 depicts this normal scaling behavior. When determining the mean of every scaling step we get a value of 0.99, which is relatively close to 1.

When taking these results into regard, we can safely say, that a normal scaling for the pine cluster is given when using the VerifierCloud with enough workload. Obviously when there are fewer verification tests to run than cluster nodes available then the scaling factor is decreased because of the lack of capacity utilization.

Pine Efficiency Scaling Test		
Number of Pines	Duration	Scaling to previous
1	3 h 25 m 31 s	n.a.
2	1 h 38 m 51 s	scaling(1,2) = 1.04
3	1 h 05m 42 s	scaling(2,3) = 1.00
4	48 m 51 s	scaling(3,4) = 1.00
5	42 m 39 s	scaling(4,5) = 0.92
6	33 m 08 s	scaling(5,6) = 1.07
7	30 m 46 s	scaling(6,7) = 0.92

Table 4.1: svcomp19–overflows executed on the Pine cluster with 1.1 GB memory limit and 750 MB Java Heap and SMTInterpol as SMT solver with 900 s time limit on Ubuntu 18.04.

4.3.2 Pine-Energy Consumption with SMTInterpol Test

Since we are using SMTInterpol as SMT solver, we need to change the CPAchecker config, so that it uses Integers as Bitvector representation. Therefore I have added the following code lines to the config files:

```
# use unbounded integers in formulas instead of int-variables.
cpa.predicate.encodeBitvectorAs = INTEGER
# use rationals in formulas instead of float-variables.
cpa.predicate.encodeFloatAs = RATIONAL
# precise handling of structs only possible with bitvectors.
cpa.predicate.handleFieldAccess = false
```

As we can see in table 4.2, the duration for each type of svcomp test differs. Since energy consumption also depends on the test duration, it also varies. Because of the unsound Bitvector to Integer conversion, some tests like the

Overflows and Reachsafety have a bad result score. The score for each individual test is calculated by svcomp19 method. For each average test row we see a tuple consisting of correct predictions, incorrect predictions and unknowns (short: (cor/inc/unk)). From these components the numerical score is calculated.

Pine Energy Consumption with SMTInterpol Test			
Test	Duration	Energy needed	Score (cor/inc/unk)
Overflows 1	29 m 02 s	10.3 Wh	-2970/691
Overflows 2	33 m 35 s	11.2 Wh	-2970/691
Overflows 3	30 m 17 s	10.7 Wh	-2970/691
Overflows avg	30 m 58 s	10.7 Wh	(243/203/5)
Memsafety 1	02 h 37 m 07 s	56.2 Wh	349/753
Memsafety 2	02 h 55 m 23 s	59.4 Wh	349/753
Memsafety 3	03 h 13 m 51 s	64.1 Wh	349/753
Memsafety avg	02 h 55 m 27 s	59.9 Wh	(251/1/223)
Termination 1	01 h 13 m 04 s	23.7 Wh	301/443
Termination 2	01 h 18 m 07 s	24.3 Wh	301/443
Termination 3	01 h 14 m 23 s	23.4 Wh	301/443
Termination avg	01 h 15 m 11 s	23.8 Wh	(177/1/72)
Reachsafety 1	10 h 12 m 50 s	200 Wh	-370/1534
Reachsafety avg	10 h 12 m 50 s	200 Wh	(448/67/384)

Table 4.2: svcomp19–overflows/memsafety/termination/reachsafety runcollection executed on the Pine cluster with 1.1 GB memory limit and 750 MB Java Heap and SMTInterpol as SMT solver with 900 s time limit on Ubuntu 18.04.

4.3.3 Apollon-Energy Consumption with SMTInterpol Test

As expected in comparison to the Pine cluster the Apollon cluster is much faster at the cost of much higher energy consumption, as we can see in table 4.3. The best test to compare both is the Memsafety avg test, since both clusters achieve almost the same score. The Apollon cluster was about 12 times faster but also needed 14 times more energy. Because of the excellent scaling value of the Pines combined with the VerifierCloud, we could also have used 12 times the amount of pines to achieve the same speed, with equally the same amount of energy used.

The Overflows Test is finished by the Apollon cluster noticeably faster since of the unsound Integer approximation for Bitvectors.

Apollon Energy Consumption with SMTInterpol Test			
Test	Duration	Energy needed	Score (cor/inc/unk)
Overflows 1	37 s	37.0 Wh	-3000/691
Overflows 2	37 s	36.4 Wh	-3000/691
Overflows 3	38 s	35.9 Wh	-3000/691
Overflows avg	37 s	36.4 Wh	(245/205/1)
Memsafety 1	14 m 53 s	869 Wh	351/753
Memsafety 2	14 m 48 s	859 Wh	351/753
Memsafety 3	14 m 49 s	857 Wh	351/753
Memsafety avg	14 m 50 s	862 Wh	(252/1/222)
Termination 1	14 m 52 s	792 Wh	311/443
Termination 2	14 m 51 s	793 Wh	311/443
Termination 3	14 m 49 s	790 Wh	311/443
Termination avg	14 m 51 s	792 Wh	(184/1/65)
Reachsafety 1	17 m 07 s	1.65 kWh	-333/1534
Reachsafety 2	17 m 17 s	1.64 kWh	-333/1534
Reachsafety 3	17 m 16 s	1.64 kWh	-333/1534
Reachsafety avg	17 m 13 s	1.64 kWh	(483/69/347)

Table 4.3: svcomp19–overflows/memsafety/termination/reachsafety runcollection executed on the Apollon cluster with 1.1 GB memory limit and 750 MB Java Heap and SMTInterpol as SMT solver with 900 s time limit on Ubuntu 18.04.

4.3.4 Apollon-Energy Consumption with MathSAT5 Test

Due to the MathSAT5 SMT solver, the overall score of all svcomp tests got better, as depicted in table 4.4. This is caused by the bitprecise encoding which is more sound than linear encoding. The paper Towards practical predicate analysis also shows this behavior [15]. Besides, the energy consumption slightly decreased, except for the Overflows Test. This one kind of test set is an exception since with MathSAT5 it runs sound compared to SMTInterpol and mostly executes the tasks correctly, why the processing time rises and approaches a similar value like on the other tests.

The Apollon cluster is again less power efficient than the pine cluster, because of the much higher energy consumption.

Apollon Energy Consumption with MathSAT5 Test			
Test	Duration	Energy needed	Score (cor/inc/unk)
Overflows 1	16 m 21 s	933 Wh	108/691
Overflows 2	15 m 05 s	861 Wh	108/691
Overflows 3	15 m 06 s	874 Wh	108/691
Overflows avg	15 m 30 s	889 Wh	(296/20/135)
Memsafety 1	14 m 51 s	837 Wh	351/753
Memsafety 2	14 m 53 s	838 Wh	351/753
Memsafety 3	14 m 51 s	840 Wh	351/753
Memsafety avg	14 m 52 s	838 Wh	(252/1/222)
Termination 1	14 m 52 s	790 Wh	311/443
Termination 2	14 m 51 s	785 Wh	311/443
Termination 3	14 m 49 s	787 Wh	311/443
Termination avg	14 m 51 s	787 Wh	(175/0/75)
Reachsafety 1	17 m 34 s	1.49 kWh	676/1534
Reachsafety 2	17 m 26 s	1.47 kWh	676/1534
Reachsafety 3	17 m 33 s	1.49 kWh	676/1534
Reachsafety avg	17 m 31 s	1.48 kWh	(472/5/422)

Table 4.4: svcomp19–overflows/memsafety/termination/reachsafety runcollection executed on the Apollon cluster with 1.1 GB memory limit and 750 MB Java Heap and MathSAT5 as SMT solver with 900 s time limit on Ubuntu 18.04.

4.3.5 Apollon-Energy Consumption with MathSAT5 and more RAM Test

In this test, we have increased the memory limit and Java Heap to determine the energy consumption difference and score increase and depict memory significance. As we can see in table 4.5 the overall score values slightly increases, due to a lower chance of getting out of memory. Since we are utilizing more RAM, the energy consumption noticeably growth.

Even with the SMT solver MathSAT5 which supports for the system better optimized features and a memory advantage the Apollon cluster has no significant higher score in Memsafety and Termination. On the other hand, there are tests like the Overflows and reduced Reachsafety which achieve viable results in contrast to the pine clusters score. This shows a lack of optimization regarding the usage of ARM-based hardware in terms of software verification. As long as the tests uses well ARM optimized libraries the pine cluster is much more efficient.

Apollon Energy Consumption with MathSAT5 and more RAM Test			
Test	Duration	Energy needed	Score (cor/inc/unk)
Overflows 1	15 m 05 s	1.10 kWh	109/691
Overflows 2	15 m 06 s	1.11 kWh	109/691
Overflows 3	15 m 06 s	1.10 kWh	109/691
Overflows avg	15 m 06 s	1.10 kWh	(297/20/134)
Memsafety 1	14 m 54 s	1.01 kWh	355/753
Memsafety 2	14 m 53 s	995 Wh	355/753
Memsafety 3	14 m 55 s	996 Wh	355/753
Memsafety avg	14 m 54 s	1.00 kWh	(254/1/220)
Termination 1	14 m 52 s	811 Wh	321/443
Termination 2	14 m 50 s	808 Wh	321/443
Termination 3	14 m 59 s	810 Wh	321/443
Termination avg	14 m 54 s	810 Wh	(179/0/71)
Reachsafety 1	25 m 08 s	2.24 kWh	724/1534
Reachsafety 2	25 m 03 s	2.25 kWh	724/1534
Reachsafety 3	25 m 04 s	2.24 kWh	724/1534
Reachsafety avg	25 m 05 s	2.24 kWh	(501/5/393)

Table 4.5: svcomp19–overflows/memsafety/termination/reachsafety runcollection executed on the Apollon cluster with 15 GB memory limit and 10000 MB Java Heap and MathSAT5 as SMT solver with 900 s time limit on Ubuntu 18.04.

4.3.6 Pine-, Apollon-Idle Power Usage Comparison

As depicted in table 4.6, the power usage per node of the Pine cluster is much lower compared to the Apollon cluster. This means, when we need less than $(17.0\,W/1.5\,W) * 168 = 1904$ Pine nodes to achieve the same performance as the Apollon cluster on verification work, we would have a lower idle energy consumption for the same performance.

Pine, Apollon Idle Power Usage Comparison			
Cluster	Duration	Power usage	Power usage per node
Pine	24 h 8 m	10.2 W	1.50 W
Apollon	8 h 12 m	2.86 kW	17.0 W

Table 4.6: Comparison of both clusters' power usage when no workload (idle state) applies.

5 Conclusion

We have shown an optimization approach to reduce the energy consumption on clusters running verification work. The used ARM processors have the potential to be ten times more power-efficient than our conventional x86 Intel cluster. Even big web service providers like Amazon AWS offer ARM-based server instances. As long as your application fully supports the ARM platform it will most likely be more power-efficient caused by the RISC-architecture using less transistor logic and lower clock rates. Therefore you need more cluster nodes to achieve the same speed performance compared to an Intel Xeon processor. Provided that the application has a good scaling behavior for the number of cluster nodes, like the VerifierCloud I have used to distribute the verification tasks; there should not be any problems caused by the lower speed performance when both cluster nodes are compared one by one.

In addition, we discussed different energy measuring approaches. RAPL is an excellent way to measure the CPU and RAM power usage but not to extrapolate the whole system power usage, since there are other computer components having a variable power usage depending on the workload. Also, it is not compatible with every CPU, especially with non-Intel CPUs. Therefore the conventional energy and power measuring with an Energy Meter was the appropriate choice for measuring the energy consumption of clusters.

When in the future even better ARM support for verification tools and libraries are given, upgrading to an ARM-based cluster could be a valid option to take into account. Especially when considering lowering the maintenance costs and ecological footprint through increasing power efficiency.

A Implementations

All project relevant code snippets are listed here chronologically in the order I have used them.

A.1 Energy Counter Value Extractor

Simple bash script to extract for the measurement important energy meter data.

```
#!/bin/sh
# This script outputs the current acc. energy consumption
# depending on OBIS code 1.8.0 (IEC 62056).
# Depending on the argument (1 or 2) you can access one of the meters data.
# The output calculation depends on the last 10 1.8.0 log entries.
# If device 1 has the latest 10 entries and device 2
# for example the 11th newest then the output result
# for device 2 is nothing (later interpreted as 0)
# because every device answers at the same frequency
# (even if nothing has effectively changed) so it means that
# device 2 is apparently offline.
# So tail argument 80 / 8 output variations = 10 "log entry timeout"

logLength=80
logPath="/var/log/vzlogger.log"

if [ $1 -eq 0 ]
    then tail -n $logLength $logPath 2> /dev/null |
    awk '($3␣~␣/mtr0/␣&&␣$5␣~/1.8.0/)␣{split($6,a,"=");␣print␣a[2]}' | tail -1
elif [ $1 -eq 1 ]
    then tail -n $logLength $logPath 2> /dev/null |
    awk '($3␣~␣/mtr1/␣&&␣$5␣~/1.8.0/)␣{split($6,a,"=");␣print␣a[2]}' | tail -1
else
    echo "ERROR:␣wrong␣parameter"
fi
```

A.2 Collectd Plugin For Energy Value Dispatching

A plugin for collectd, written in python to dispatch the energy meter values to other applications.

```
#!/usr/bin/python

import collectd
import os, subprocess

get_energy_count_path = "/home/pi/Energy_Counter/scripts/get_energy_count.sh"
meter_id_0 = "0"
meter_id_1 = "1"


def retrieve_energy_count(id):
    """Reads the current energy count values for a energy meter from the logfile via
    bash script, this script returns the latest value for each id
    (depending on the last 10 values logged)
    """
    cumulative_energy_count = subprocess.Popen(["bash", get_energy_count_path , id],
    stdout=subprocess.PIPE).communicate()[0]
    if cumulative_energy_count == "":
        cumulative_energy_count = 0.0
    else:
        cumulative_energy_count = float(cumulative_energy_count.replace("\'","\\n"))
    return cumulative_energy_count

def read_func():
    """Retrieves for both energy meter their current values and adds them together
    Then dispatches the values for collectd
    """
    cumulative_energy_count_combined = retrieve_energy_count(meter_id_0) +
    retrieve_energy_count(meter_id_1)

    val = collectd.Values(type='energy')
    val.plugin = 'energy_count'
    val.dispatch(values=[cumulative_energy_count_combined])


def main():
    collectd.register_read(read_func)

main()
```

A.3 Vzlogger Log To CSV Converter

This script allows an easy way to export all gathered data into a csv file.

```
#! /usr/bin/python
# -*- coding: utf-8 -*-
import pytz, sys, csv
from datetime import datetime
from tzlocal import get_localzone

meter_id = "mtr1"
obis_code = "1-0:1.8.0"
logfile_path = "/var/log/vzlogger.log"
outfile_path = "out/log"
outfile_format = ".csv"
date_format = "%Y-%m-%d_%H:%M:%S"
value_id = "value="
timestamp_id = "ts="

def convert2csv(line):
    """Converts the log file into csv, depending on the given obis code
    and meter id. The datetime and energy value will be extracted for
    the given line and then returned in csv style
    """
    if meter_id in line and value_id in line and obis_code in line:
        new_line = line.split("_")
        time = int(new_line[6].replace(timestamp_id,"").replace("\n",""))
        date = datetime.fromtimestamp(time / 1000, get_localzone()).
        strftime(date_format)
        millis = str(time % 1000)
        value = new_line[5].replace(value_id,"")
        return date + ":" + millis + "," + value + "\n"
    else:
        return ""

def reverseCSV():
    """Reverses a csv-file, in terms of line ordering
    """
    with open(outfile_path + "_" + meter_id + "_reversed" + outfile_format) as fr,
    open(outfile_path + "_" + meter_id + outfile_format,"wb") as fw:
        cr = csv.reader(fr,delimiter=",")
        cw = csv.writer(fw,delimiter=",")
        cw.writerow(next(cr))
        cw.writerows(reversed(list(cr)))

def convertFile():
    """Reads the given logfile linewise and
    converts each line to a csv compatible line
    regarding on the timestamp and energy value
    """
    log = open(logfile_path, "r")
    out = open(outfile_path + "_" + meter_id + "_reversed" + outfile_format, "w")
    out.write("time,value\n")
    for line in log:
        out.write(convert2csv(line))
    out.close()
    log.close()
    reverseCSV()

def interpreteArguments():
    """Very simple argument interpreter
    This method does not validate the arguments.
    """
    global meter_id, obis_code, logfile_path
    if len(sys.argv) > 4:
        print("Too␣many␣arguments")
        print("<1.␣meter␣id>␣<2.␣obis␣code>␣<3.␣logfile␣path>")
        print("<1.␣meter␣id>␣=␣mtr0␣|␣mtr1")
        print("<2.␣obis␣code>␣=␣1-0:1.8.0␣...␣1-0:2.8.2")
        print("<3.␣logfile␣path>␣=␣/path/to/logfile.log")
        print("Possible␣combinations:␣1.␣or␣1.␣and␣2.␣or␣1.␣and␣2.␣and␣3.")
        sys.exit()
    if len(sys.argv) >= 2:
        meter_id = sys.argv[1]
    if len(sys.argv) >= 3:
        obis_code = sys.argv[2]
    if len(sys.argv) >= 4:
        logfile_path = sys.argv[3]

if __name__ == "__main__":
    interpreteArguments()
    convertFile()
```

A.4 Verifiercloud Addition

Alongside with other small additions in ProcessReader.java and LinuxSystemInformationProvider.java in the VerifierCloud software, I have added a new getCPUModel() method in ProcessReader.java to obtain all relevant CPU model information also for ARM processors.

```
public String getCPUModel() throws SystemEnvironmentException {
    try {
      FailingProcessExecutor pe = new FailingProcessExecutor(logger, "lscpu");
      pe.join();
      String rawCpuInfoLines = pe.getOutputRAW();
      List<String> cpuInfoLines = Arrays.asList(rawCpuInfoLines.split("\n"));

      if (!pe.getErrorOutputRAW().isEmpty()) { //fallback to old cpuInfo determination
        cpuInfoLines = Files.readAllLines(CPUINFO_PATH, Charset.defaultCharset());
      }
      return getCleanCPUModel(cpuInfoLines);
    } catch (IOException e) {
      throw new SystemEnvironmentException(e);
    } catch (Exception e) {
      throw new SystemEnvironmentException("Unable to determine energy usage.");
    }
  }
```

Bibliography

[1] G. Calandrini, A. Gardel, I. Bravo, P. Revenga, J. L. Lázaro, and F.J. Toledo-Moreo. Power measurement methods for energy efficient applications. In *Sensors*, 2013.

[2] A. Crane. Modernizing and protecting the electricity grid. In *The Bridge - Linking Engineering and Society*, page 3, 2010.

[3] Bundesministerium für Energie und Wirtschaft. Energieträger. In *Energiedaten: Gesamtausgabe*, page 38, 2018.

[4] Bundesamt für Sicherheit in der Informationstechnik. Teil b: „SML – Smart Message Language“. In *Technische Richtlinie BSI TR-03109-1*, 2013.

[5] J. Ganz, M. Beyer, and C. Plotzky. Time-series based solution using influxdb. 2015.

[6] M Haehnel, B. Doebel, M. Voelp, and Hermann Haertig. Measuring energy consumption for short code paths using rapl. In *ACM SIGMETRICS Performance Evaluation Review*, 2012.

[7] A. Menditto, M. Patriarca, and B. Magnusson. Understanding the meaning of accuracy, trueness and precision. *Accreditation and Quality Assurance*, 12(1), 2007.

[8] EMH metering. Produkthandbuch für den elektronischen Haushaltszähler ED300L nach FNN Lastenheft EDL. 2010.

[9] A. Mueen, E. Keogh, Q. Zhu, S. Cash, and B. Westover. Exact discovery of time series motifs. In *SIAM international conference on data mining*, 2010.

[10] S. Naqvi and S. Yfantidou. Time series databases and influxdb. Research project, Université Libre de Bruxelles, 2017.

[11] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems. 2001.

[12] A. Rafindadi and I. Ozturk. Impacts of renewable energy consumption on the german economic growth: Evidence from combined cointegration test. In *Renewable and Sustainable Energy Reviews Volume 75*, pages 1130–1141, 2017.

[13] European Standard. Electricity metering equipment (a.c.) - particular requirements - static meters for active energy (class indexes a, b and c). In *EN 50470-3:2006 European Standard*, 2006.

[14] N. Steinger. Measuring, visualizing, and optimizing the energy consumption of computer clusters. Bachelor thesis, Universität Passau, 2017.

[15] P. Wendler. *Towards Practical Predicate Analysis*. PhD thesis, Universität Passau, 2017.

Declaration of Authorship

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references.
This thesis was not previously presented to another examination board and has not been published.

Munich, 5.11.2019

. . . .
Maximilian Hailer