University of Zimbabwe

Process Level Evaluation of Energy Efficiency of Virtualised Databases in IoT cloudlets

A thesis submitted to the cluster Computer Science Education in partial fulfillment of the requirements for the award of a

Master of Science Degree

(Computer Science)

By

Solomon H Kembo

Supervisor: Dr. Kuda Dube

July 2018

Acknowledgements

I would like to express my heartfelt gratitude to my supervisor, Dr. Kuda Dube for his valuable guidance, patience and encouragement throughout the period of the thesis work. His attention to detail and quality made it possible to compile the thesis document and running the experiments on schedule.

I would also like to thank the chairman of the Computer Science department Dr. Benny Nyambo for his insightful comments and encouragement.

On the whole, I would like to thank the Department of Computer Science for this educational opportunity which has tested and pushed me beyond my abilities.

Lastly, I would like to thank my parents, siblings and friends for all the support.

Abstract

Cloud computing has been the bedrock of data storage for most IoT applications. However, cloud computing in developing communities is costly, suffers from latency bottlenecks and raises a number of privacy concerns. Consequently, an alternative paradigm for data management in the form of cloudlets is widespread in developing communities.

A prominent use case of cloudlets is the use of single board computers on a local network similar to edge gateways. However, challenges of grid electricity make cloudlet deployments in poor and remote communities a huge challenge. Optimising for power in software engineering is also in its infancy owing to lack of mature methodologies and tools to measure energy.

This work develops testbeds to measure the impact of power consumption on constrained cloudlets of three databases, namely InfluxDB, Postgres and MongoDB on a Raspberry Pi acting as an edge gateway. The study uses PowerAPI software-defined power meters to measure the power consumption of the databases. The databases, in the experiment, store data from a python time series application using HTTP web APIs and python APIs packaged as Docker containers.

The results of the experiments show that Postgres is the most energy-efficient of the databases and that using Docker containers and HTTP APIs has an insignificant effect on the power consumption of the Raspberry Pi.

Table of Contents

List of Figures

List of Tables

Appendices

List of Abbreviations and Symbols

CoAP	Constrained Application Protocol
OO	Object Oriented
MQTT	Message Queue Telemetry Transport
MEC	Mobile Edge Computing
SoC	System on a Chip
IoT	Internet of Things
SBC	Single Board Computer
API	Application Programming Interface
HTTP	Hypertext Transport Protocol
SQL	Structured Query Language
LEAD	Latency awarE workloAd offloaDing
DSDV	Distance-Sequenced Distance Vector
CPU	Central Processing Unit
GPU	Graphics Processing Unit
DRAM	Dynamic Random Access Memory
SSD	Solid State drive
TEEC	Tool to Estimate Energy Consumption
HPC	Hardware Performance Counter
MSR	Model Specific Register
RAPL	Running Average Power Limit
EACOF	Energy-Aware Computing Framework
LTE	Long Term Evolution
REST	Representational State Transfer
AAL	Ambient Assisted Living
RFID	Radio Frequency IDentification
TDP	Thermal Design Power

Keywords: Energy Modelling, IoT, Cloudlets, Databases, Power Meters

1 Introduction

Internet of Things (IoT) is characterized by small gadgets such as sensors and actuators that operate in very constrained environments with regards to power and processing capacity. A number of studies in the domain of energy optimisation such as (Mun, Dinh, & Kwon, 2016) investigated the workings of IoT protocols including Constrained Application (CoAP), WebSocket and Message Queue Telemetry Transport (MQTT) with the goal of assisting IoT applications programmers select the most energy-efficient protocol

Whilst efforts to optimize for power in the low-level IoT protocol implementations exist, cases of power optimization in the upper layers of the IoT stack, that include data aggregation and storage, are far and wide. There is therefore a need to investigate ways of measuring and optimising for power consumption within the higher levels of the IoT stack in order to optimize IoT applications for power.

According to (Devon M. Simmonds, 2012) software development has evolved from imperative, low-level machine code to declarative, High-level Object-Oriented (OO) and functional programming. This evolution shows a progression from an orientation on physical electronics to a focus on high-level mathematical, abstract models and implementations.

A similar progression is also taking place in the optimisation of energy in computing. Enhancements in energy started off at energy sources which include the electrical grid and alternative energy sources such as the battery. Further optimizations were then witnessed in the electronics and computer hardware layer. Optimisations in hardware are evident in the processors and data centres (Combarro, Tchernykh, Kliazovich, Drozdov, & Radchenko, 2016).

Lately, the progression has reached the software layer, a layer that is more mathematical and abstract, almost similar to the software development paradigms' evolution. (Hindle, 2016) suggests that the late entry of software into energy measurement and optimisation can be attributed to the fact that electricity and hardware are much closer to natural sciences, whereas software engineering has elements of social sciences. Natural sciences are known to rely on measurements and empirical data while social sciences are subjective in nature. The subjective nature of software engineering has therefore made measurement of power consumption a challenge.

Isolated software power optimizations efforts have been observed in mobile applications development. However, only a handful involved lightweight techniques that were intuitive to software such as proposals made by (Hao, Li, Halfond, & Govindan, 2013).

The evident progression from physical layer power optimizations towards optimizations in software presents an opportunity to study ways that software development can intuitively recognise energy efficiency of applications as a worthy design goal. This is made possible by creating software that is able to measure power consumed by constituent software components. Such granular measurement of power by energy-aware applications must take into account differences in hardware, operating systems, environments and software domains when developing energy models.

1.1 Motivation

This section summarises motivating opportunities in the field of computing that can be leveraged to complement existing energy-saving initiatives. Factors motivating this study include the predictive evolution of the battery, continued miniaturization of printed circuit boards, alternative low latency data processing architectures to the cloud model as well as the use of software design in reducing energy consumption.

Evolution of the Battery and System-on-a-Chip

According to Koomey's Law, the number of computations per joule doubles every 1.57 years. This implies that battery requirements will fall by a factor of 2 every one and half years (Koomey, Berard, Sanchez, & Wong, 2011) . This rate is steeper than Moore’s Law and has been fairly consistent since the 1950s. This study is in part motivated by the need to leverage the on-going and consistent enhancements of the battery as predicted by Koomey’s Law. The accessibility of the battery will allow low technology applications to be powered off the grid as grid electricity has not been cost-effective in constrained applications and developing communities.

Recently the semi-conductor industry has witnessed the disruptive potential of System-on-a-Chip (SoCs) in the post-PC era. During the PC-era, form-factor of the processor was rarely a consideration but the advent of the smaller devices such as mobile phones and tablets has given form-factor more relevance. Another important factor that has become relevant is power-per-cost. These positive developments in the semi-conductor industry give credence to investigating power optimisations of SoC-powered IoT applications such as those deployed on single-board computers.

Multi-access Edge Computing Developments

Previously known as Mobile Edge Computing (MEC), Mult-access Edge Computing presents developers with a myriad of cloud computing opportunities in ultra-low latency environments. This is evidenced by the evolution of new use cases in vertical business segments involving fields that include location services, Internet of Things, augmented reality as well as optimised local content distribution that is cached. MEC essentially aims to solve the challenge of increased transmission latency caused by the increase in the physical distance between the user and the cloud.

The most predominant data storage architecture in the field of IoT involves persisting collected data in the Cloud. This is attributable to the fact that cloud computing efficiently pushes storage and computation away from the operations that collect the data. However, according to (Zhu, Leung, & Shu, 2015), in the context of Internet of Things, cloud computing has several limitations when factors such as latency, the privacy of data and a need to control devices in near-real time are considered.

The increased adoption of cloudlet architectures, in which limited cloud computing services are available on the edge of the network, presents IoT opportunities for resource-constrained areas such as Africa and other developing communities. Cloudlets allow for the implementation of MEC using commodity tools.

A number of initiatives to study the use of constrained edge gateways in cloudlets for applications that traditionally required the cloud are on the rise. A domain that is gaining traction in the use of edge gateways to host applications is data storage in the form of databases.

Improved Persistence of Databases in Docker Containers

The concept of process isolation started with the chroot system calls, in the 1970s, which changed the root directory of a process together with its child processes to different locations. The concept of containers has since evolved significantly with widespread use peaking with the introduction of LinuX Containers (LXC) which utilised cgroups and Linux namespaces within the Linux kernel. The introduction of Docker containers in 2013 increased the popularity of containers.

Previously, docker containers were essentially ephemeral lightweight virtual machines, ideal for stateless microservices-based applications. This meant databases, because of their need to persist data, were not suitable for Docker containers. However, development of volumes in Docker containers allows for databases to be virtualized within Docker containers.

Green Software Engineering

Sustainability is fundamental to achieving the Millennium Development Goals in eradicating poverty. Sustainable Information Technologies in the form of Green IT offer an opportunity to enable energy saving through computing resources, particularly computer hardware.

A reference model and suitable methodologies have been lacking in previous attempts to implementing green software engineering. GREENSOFT provides numerous possibilities for Green-by-IT by providing a promising reference model that guides the use of software in efficient energy use.

The advent of Green IT requires developers to consider the energy footprint of applications in order to strike a healthy balance of application development performance factors such as performance and security.

The developments in the computing industry, cited above, provided the impetus to explore them further and capitalise on them in this research study.

1.2 Problem Statement

The constrained nature of Internet of Things applications requires energy efficiency particularly in off-the-grid use cases. However, most of power optimization efforts in computing are confined to hardware, particularly processors (Babu, 2014) and servers in data centres (Combarro et al., 2016).

A possible reason why research in optimizing software is not widespread might be explained in the difficulties in isolating and attributing power usage to individual software components for the purpose of measurements. Measuring power usage of hardware components using physical components like multimeters is intuitive and accurate. However, it has limitations when measuring software components because the measurements are not granular but coarse-grained.

(Hindle, 2016) also highlights the challenges of coming up with generalizable energy models. It is currently difficult to model software energy consumption as the applications will be running on diverse hardware, operating systems, runtime environments and software domains

Another plausible explanation to the scarcity of software-based power optimisation efforts is lack of empirical data to convince software developers on the benefits of optimizations of power (Manotas et al., 2016) . Unless there are power measurement and optimization metrics it is difficult for developers to ascertain whether optimizing for power warrants the resultant performance and security trade-offs.

Lack of granularity in physical hardware measurements, unavailability of generalizable platform-agnostic energy models and lack of empirical data to inspire IoT software engineers to consider energy consumption during software design are challenges inhibiting the progress of Green Software Engineering. There is therefore a need to provide solutions through rigorous research studies to ensure sustainable software development.

1.3 Objectives

The ultimate goal of this study is to provide contextual and empirical evidence of the need to measure and optimise for power during the design of IoT applications. In order to effectively achieve the research goal a number of clear objectives have been identified as follows:

1. To determine if power consumption efficiencies are significant enough to warrant the resultant performance, functionality and security trade-offs in IoT applications.
2. To provide fine-grained energy consumption metrics for the cloudlet IoT architecture at the data storage layer using hardware-agnostic tools.
3. To develop a decision-making framework for IoT developers based on energy efficiency of virtualised databases within cloudlet environments that utilise Single Board Computers as edge gateways.

1.4 Contributions and Significance

The study will provide software-defined power meters and metrics to aid IoT software developers of virtualized, cloudlet IoT applications in selecting the most energy-efficient database management setup in constrained cloudlets. This will be achieved through process-level comparison of fine-grained power consumption metrics of leading database management systems.

The metrics will allow researchers to compare the energy efficiency of three different types of databases storing time-series data. The set of recommendations from the metrics produced provide a decision-making framework to IoT developers on the most energy-efficient data storage for cloudlet-based IoT application development.

The software-defined power meters can be reused in applications as they are delivered as automated pipelines collecting data from the outside world through a serial-connected microcontroller. Spawn systemd-controlled power meters measure the power consumption and store the collected values in a mass deployment database, SQLite, for further analysis.

To facilitate the reproducibility of this study the artefacts will be availed to the IoT research community on a gitlab repository: https://gitlab.com/solkem/point. The artefacts availed include:

- The systemctl and bash scripts used to compute the energy consumption using PowerAPI,
- The python time-series application and energy data from the SQLite database.
- The time series data collected during the experiments from Postgres, InfluxDB and MongoDB within and without Docker images and containers.

Given the constrained nature of cloudlet deployments, software developers can utilise the output of this study as an intuitive mechanism to evaluate the energy an IoT application consumes. The study also aims at contributing to the discussion on the readiness of virtualised databases on constrained IoT devices and networks with regards to energy efficiency based on the testbed metrics.

1.5 Outline of Thesis

The rest of the thesis is structured as follows. Chapter 2 presents the background of the study as well as the contextual limitations of the IoT use case in which the study was undertaken. A review of relevant literature and related work in energy measurement and optimisation efforts follows in Chapter 3. Chapter 4 outlines the methodology utilised to get to the solution of the research problem. Experiments conducted to observe the energy consumption of data storage using the PowerAPI middleware toolkit are presented in Chapter 5. This is followed by results and analysis of the observed phenomenon in Chapter 6. A discussion of the results follows in Chapter 7 and concluding remarks are drawn in Chapter 8, which also provides directions for future work.

2 Background and Context

This section explores the prevailing IoT state-of-the-art and use cases. Emphasis is placed on the constraints imposed by the dominant yet unsustainable IoT models that require robust infrastructure to be in place. An understanding of the African computing context will reveal challenges in effectively deploying IoT solutions.

It is important to understand the current state of computing in general and IoT architectures in particular, in developing countries in order to prescribe solutions that are relevant to the context. We explore the state of cloud-based deployments in developing countries.

2.1 High Costs of Cloud Computing

The most common IoT models consist of local networks made up of embedded systems that offload collected data to cloud platforms for further processing and provision of value added services such as analytics. In resource constrained communities the costs of cloud services including AWS and Micosoft Azure are unsustainable in IoT projects. This has been corroborated by the UNCTAD report (Fredriksson et al., 2013). Survey respondents from Ghana identified cloud computing challenges that include reduced control of data, the risk of inaccessibility of essential cloud services, high costs of communication, breach of data security and privacy, and migration costs as well as vendor lock-in due to limited interoperability.

2.2 Latency Bottlenecks

Another challenge affecting IoT deployments relates to issues of latency. Physical network latency problems have profound negative effects on IoT control and monitoring systems that require near real time access to data. (Abrahamsson et al., 2016) identifies lack of contextual understanding when deploying cloud computing solutions as studies reflected the state of cloud computing from the point of view of western countries. The study states that the existing cloud computing viewpoint is infeasible in developing countries primarily due to unreliable and slow internet connections.

(Johnston & Cox, 2017) cite the limitations of the cloud architecture in applications that require real-time responses. Factors cited as lowering adoption of cloud computing in low and middle-income countries include: quality and availability of cloud-based infrastructure, high costs and lack of legal and regulatory frameworks to deal with privacy concerns

2.3 Privacy Concerns

Privacy concerns relating to storage of personal data in cloud services have escalated recently. Effects of personal data privacy breaches are acute in domains that collect sensitive data such as in healthcare (Jeyanthi, Thandeeswaran, & IGI Global, 2017) . The study, which focused on Ambient Assisted Living (AAL) in healthcare, identified security and privacy concerns attributed to insecure authentication mechanisms, issues in lightweight protocols, cloud computing and difficulty in using RFID technology.

Even though cloud computing has witnessed phenomenal adoption rates in developed countries, the adoption rates in developing countries have been low as a consequence of high costs, high latency and lack of security and privacy. Consequently, a number of research studies exploring an alternative localised cloud computing known as cloudlets have flourished, headlined by work at Carnergie Mellon University's Software Engineering Institute (Satyanarayanan et al., 2013)

2.4 Alternative Cloud Storage Solution: Edge Computing and Cloudlets

A number of studies targetting developing communities have investigated the use of cloudlets as an alternative data storage solution. Below is a presentation of the most prominent cloudlets studies.

(Satyanarayanan et al., 2017) uses local cloudlets to analyse data from always-on, bandwidth-intensive video cameras whilst pushing extracted indexed information and meta-data to the cloud. Using Gigasight software the study is able to preserve privacy in the videos through a process of denaturing of streams within virtual machines.

(Abrahamsson et al., 2013) explored the concept of micro-data servers in developing communities. The study justifies a cloudlet architecture in community-based services as a way to improve availability in cases of power outages, lower costs associated with cloud computing and ensuring privacy. The micro-data centre developed in the study behaves as a connector of clients to the cloud as well executing community-based services on the local infrastructure. The cloudlet setup deployed contains four main attributes. It utilises low-cost components that are easily available, runs on solar and batteries for long periods of time, was designed to withstand harsh conditions such as high temperatures and extensively uses open platforms and standards.

(Paethong, Sato, & Namiki, 2016) asserts that data is the most important component of an IoT application and proposes the use of a raspberry Pi running a low-power distributed database as part of a middleware solution.

(Morabito, 2017) utilised edge computing in lightweight virtualization also referred to as device virtualisation within resource-constrained IoT deployments. The study evaluated Raspberry Pi and Odroid Single Board Computers (SBCs) deployed on the edge of the network. The experiments empirically demonstrated an opportunity of efficiently deploying SBCs as edge devices.

Through utilising geographically distributed cloudlets in mobile networks, (Sun & Ansari, 2017) demonstrate how the cloudlet architecture improves Quality of Experience for Mobile Users by lowering the average response time. The study involved attaching cloudlets to Base Stations providing powerful computing to local mobile users. The proposed Latency awarE workloAd offloaDing (LEAD) strategy achieved lower response times by offloading computing requirements to suitable cloudlets on the mobile edge.

(Mthunzi, Benkhelifa, Jararweh, & Al-Ayyoub, 2017) propose a cloudlet solution in digital forensic investigations. The solution enables investigators to maintain a comprehensive evidence trail by ensuring that cloudlets remain the sole custodians of data centres. The study reveal the challenges of data stored on the cloud that include lack of control by investigators. The cloudlet solutions proposed, cater for the four phases involved in the digital forensic investigation process. As such four different services including: Investigation, Evidence Consolidation, Case Management and Evidence Ingestion Cloudlets are deployed

A Distance-Sequenced Distance-Vector (DSDV) is used to evaluate the performance of a cloudlet-based approach for an ad-hoc model for mobile computing based cloud computing by (Al-Rousan, Al-Shara, Jararweh, & Alshayeji, 2017). Different node speeds where measured for various workloads that were offloaded to the cloudlets. The results showed that passing workloads to cloudlets was much faster that using the enterprise cloud.

2.5 Challenges of Cloudlets in Developing Countries

The greatest benefit of traditional cloud computing is a transfer of operational management to the cloud computing operator. This advantage of cloud computing is apparently a drawback of the cloudlet architecture. Within cloudlets, resources have to be managed internally by the owners of the cloudlet. This introduces new devops challenges that are not usually associated with the cloud computing paradigm.

1. Energy Efficiency on Constrained Cloudlets

Most IoT deployments in developing communities add value to remote settlements that benefit from cyber-physical systems in the agriculture and healthcare. However, these challenged areas are not on the electrical grid or the electricity is not reliable. Consequently, optimising energy consumption is essential for the success of IoT deployments. Energy optimisation in the field of ICT is still in its infancy as evidenced by lack of standards in energy measurement and optimisation.

A challenge for software developers willing to factor in power consumption in applications is the lack of methodologies that enable experiments to be replicated . The general myth that power ICT optimizations exist only in hardware optimizations or renewal energy sources has also decelerated the adoption of Green Software Engineering according to (Lago, 2015) . Therefore there is a need for reproducible research work on the merits of software power optimizations and simplified yet rigorous standards for measuring that can debunk the “hardware only” myth.

Unlike in the field of physical sciences, energy consumption measurements have not been widely adopted by the software engineering community. (Hindle, 2016) identifies reasons for the lukewarm adoption as: difficulties in attributing energy usage to specific software elements, isolation of components as well as replication of experiments. Complications arising from Cloud Computing which uses virtualization technologies are also cited as factors that discourage software professionals from measuring software energy consumption.

2. Difficulties in Attribution of Energy Usage

Most efforts to measure energy consumption operate at a very low level of granularity such as measuring at the processor and operating system level (Dong & Zhong, 2011) . An alternative approach of estimating energy consumption, similar to that of (Hao et al., 2013) , that uses simulators, has the drawback that simulators usually run a thousand times slower than underlying hardware. Consequently, hardware-based methods of estimating or measuring energy consumption are not cost-effective and are too complex for a software developer.

3. Lack of Energy Consumption Tools and Metrics

Another challenge in energy consumption measurements is the software engineers' indifferent perspective on energy measurements. (Manotas et al., 2016) interviewed eighteen (18) software practitioners at Microsoft and surveyed a total of four hundred and sixty-four (464) software developers at ABB, IBM, Google and Microsoft. The study revealed that 80% of developers were willing to trade-off energy efficiency for better performance and security. Half of the respondents believed that energy usage affects software design but only 40% traditional developers said energy usage was part of their requirements. These responses explain the lethargy of the software engineering community in taking up energy measuring and optimization.

4. Energy Efficient Data Processing on Constrained Edge Gateways

Database management is a key cloud computing component within traditional IoT deployments due to the huge volumes, velocity and variety of data. The use of the cloudlets requires that data management be managed on the edge of network preferably on the gateway. This poses a new challenge as gateways used in most cloudlets are constrained in size. A number of deployments have utilised single-board computers as gateways within cloudlets (Morabito & Benchmark, 2016), (Elkhatib et al., 2017), (Abrahamsson et al., 2013), (Priyanka, 2016), (Kortas, Azzabi, Arbia, & Tahar, 2015), (Paethong et al., 2016), (Constant, Borthakur, Abtahi, Dubey, & Mankodiya, 2017), (Morabito, 2017).

A number of database vendors have responded to IoT data needs by introducing IoT focused features that include

- embedded processing,
- specialised indexing,
- in-memory processing,
- time series focus,

- support for microservices and
- web API exposure.

Whilst these IoT related features are ideal for the growth of cloudlets very few database vendors factor the need to optimise for power consumption which is essential in a cloudlet setup.

2.6 Case Study: IoT Aquaponics Project at St Peters Mbare in Zimbabwe

A case study of a school that is inspiring students to take IoT through the notion of Makerspace in Zimbabwe is studied to understand fully the school's cloud and internet related challenges.

The Makerspace at St Peters Mbare developed an Aquaponics project that is monitored by an IoT application. The system uses an Arduino board to measure the environment through water-level, temperature, pH and superconductivity sensors. It also has a fish feeder to automate the process of feeding fish. The data collected by the sensors is saved in a MongoDB database hosted on a Raspberry Pi 3 computer. The Raspberry Pi is connected through WiFi to the school's network. The project has limited use of the internet or cloud computing due to high connectivity costs.

The aim of the project was primarily to teach technology to students. The school is in the process of creating a knowledge repository reflecting experiences with the system. Since the project is not core to the school activities an additional costs associated with the project threaten its continued existence. The school is therefore exploring ways to reduce costs associated with the project. Cutting down on the energy footprint is one avenue the school is exploring.

In response to the need to find energy efficient alternatives this study aims to understand the effect of having an application of the magnitude of a database on a constrained off-the-grid IoT network like the one at St Peter's Mbare IoT Makerspace.

3 Literature Review

A review of previous and related work in power measurement and optimization is conducted with the aim of critically analysing the past studies and understanding the core issues around energy-aware software applications. Central to the study are issues relating to:

- challenges in measuring power in software,
- on-going green software engineering efforts,
- development of hardware-independent energy models and
- modern power measurement tools in an IoT cloudlet architecture utilizing a Raspberry Pi as an edge gateway.

3.1 Literature Review Framework

Appendix I illustrates the conceptual framework on which this study is founded on. It shows the process adopted in identifying the research gaps from related, published work within the domain of optimising energy consumption within IoT cloudlets. The conceptualisation of the research problem and the selection of the most suitable research tools allowed for the development of an optimal scientific methodology to solve the research problem.

This study is centred on the concept of benchmarking energy efficiency of common database storage systems within an Internet of Things use case. In the process of surveying the research domain, a guiding conceptual framework connecting a number of contexts, theories and concepts progressively coalesced. The framework covers the following concepts: exploring means of measuring power, identifying the ideal level or granularity of measurement, establishing cost-effective, generalisable and accurate energy measurement tools, as well restricting the study to pertinent areas within the computer science domain that require energy optimisation the most.

3.2 Software Defined Power Meters

Consideration by software engineers of energy consumption in software development requires consistent methodologies for correct and effective measurement of energy. Traditionally, power measurement required physical tools that include multimeters and specialised integrated circuits. The use of hardware in energy measurement results in accurate measurements. However, using hardware for measurements makes it difficult to identify energy inefficient parts of the system because hardware power meters are incapable of attributing energy consumption to individual units of the system.

Cloudlet architectures are a response to costs associated with cloud computing. Another reason that has justified the use of software, which is mostly free and open source, is that hardware power meters are generally expensive. The need for cost-effective and fine-grained measurements adds credence to exploring alternative ways of power measurement besides using hardware.

An alternative to physical meters is the use of software defined power meters. Use of power meters is dominating energy optimisation studies within the computing domain. Cost-effectiveness and flexibility afforded by software power meters made them a natural choice for this study. Using software-defined meters, three levels of granularity are available, and they include node, process and source levels.

3.3 Process Level Granularity

The goal of this study to to provide IoT developers with evidence of the energy efficiency of data storage software that has been moved away from the cloud onto the edge of the network. This evidence should ideally become a decision framework when choosing the most energy efficient data storage software. Node level energy measurements allow developers to measure and optimise the operating environment of the application. However, such system level data might not be useful to developers optimising high-level applications.

Source level energy optimisation provides developers with the highest level of granularity when optimising for power. The focus of this study is However, on how to operate data storage software that is traditionally deployed on the cloud. Focus on databases limits the use of source level granularity as some useful databases are not open source thus the developer does not have access to the source. The scope and timeline of the study does not allow for source level measurement because it requires the developer to have intricate knowledge of the workings of the database in order to be the most effective.

Given the time constraints and a need for a fair level of granularity, process level measurements become the ideal level of measurement.

3.4 Power Measurement Domains and Targets

Software modelling of power relies on observing the stress applied to different computational units within a system. Whilst most of the software modelling studies focus on the CPU a number of researches exploring other components within the computer system, including the GPU, are on the rise.

3.4.1 GPU Modelling

Differences between the GPU microarchitecture and CPU architectures is a reason why few GPU-based software models are available in the research community according to the work of (Lim & Lakshminarayana, 2013). Using the McPAT modelling tool the study managed to estimate consumption of the Fermi GPU within margins of error of 7.7% for microbenchmarks and 12.8% for merge benchmarks.

(Qiyao Xie et al., 2012) developed an analytical GPU power model based on the FERMI GPU architecture using architectural and application parameters as inputs. The model that was developed in the study, with an accuracy rate of 90%, was able to attribute power consumption to the nVidia Fermi GPU as well as memory.(Kim, Kim, Kim, Sung, & Chung, 2015) argues that the adoption of adjustments of power and speed settings using Dynamic Voltage and Frequency Scaling in smartphones requires accurate power models to quantify energy efficiency.

Utilising implementations of Fourier transform (Ukidave, Ziabari, Mistry, Schirner, & Kaeli, 2014) optimised the GPUs of systems made up of diverse components such as SoCs, APUs and processors. The consequent optimisations resulted in a reduction of power of 27% and an increase in performance speed of a factor of 1.8. The study was inspired by the adoption of the OpenCL language which targets programming on heterogeneous systems.

Advances in the GPU technology are mostly focused on improving performance which explains the influx of studies that include (Stokke, Stensland, Halvorsen, & Griwodz, 2016), (Bridges, Imam, & Mintz, 2016), (Hosseini, Peters, & Shirmohammadi, 2013), (Song, Su, Rountree, & Cameron, 2013)

3.4.2 DRAM Modelling

A number of advances have been witnesses in the internal memory domain, the most notable being Solid State drives. However, most advances come with an energy cost. (Shey et al., 2016) hypothesises that energy and time models significantly differ among numerous between manufacturers' designs. Using the TRIM commands, the study develops an analysis technique using three different types of SSDs.

The extensive work of (Acar et al., 2016) uses dynamic power constituents: activate, precharge, read and write power to model the power consumption. Using Tool to Estimate Energy Consumption (TEEC), the study utilises parameters supplied by manufacturers and Java agents to develop instrumentations of the application. The study also optimised code for energy using code optimisation techniques that include eliminating of common expressions, code motion, strength reduction as well as unrolling loops.

3.4.3 CPU Modelling

Most of the software energy modelling is targeted at the CPU as evidenced by an extensive array of CPU-oriented power optimization methodologies, tools and techniques. In surveying research direction in power optimizations (Reda & Nowroz, 2012) characterized CPUs into three classes including general-purpose processors, system-on-chip embedded systems processors and field programmable gate arrays.

A number of energy optimization studies have utilised Hardware Performance Counters (HPC) to estimate power consumption of CPUs. Common HPC such as fetch, miss/hit, and retired instructions counters. Through extending the Performance API (PAPI) (Weaver et al., 2012) applied a one-sided factorization of dense matrices to establish whether the PAPI library can provide energy measurements using existing interfaces.

3.4.3.1 Vendor-Specific Modelling: Intel RAPL

RAPL is a software power model that microprocessor maker Intel uses to expose energy meters through Model Specific Registers (MSR). (Dongarra, Ltaief, Luszczek, & Weaver, 2012) extended PAPI associated with RAPL together with the hybrid meter PowerPack. The study revealed a significant accuracy from the software models developed by RAPL. A number of research studies (Gallo, Chichizola, & Giusti, 2018), (B, He, Cao, & Kondo, 2018), (Marcus, Marcus, & Hermann, 2012), on the energy efficiency of most High Performance Computing environments have model power consumption using RAPL.

In addition to the RAPL features most Intel processors have standard energy saving mechanisms. The most common methods include TurboBoost, C-States, SpeedStep and Hyper-Threading. TurboBoost enhances the performance of the CPU by automatically increasing the frequency, beyond the Thermal Design Power (TDP), in the event of the operating system requiring higher performance states of the processor. It is also referred to as P-state. The increased performance will consequently increase power consumption. However, users have the flexibility of disabling the feature if energy is a priority over performance.

3.4.3.2 Static and Dynamic Energy

Attributing energy to software components requires an understanding of the types of energy existing within a computer system. Energy that a computer consumes can generally be classified into static or fixed and variable or dynamic (Orgerie & Lefevre, 2014). Static energy is when energy consumed is dependent on the system type and does not change. Dynamic element is the consumption that is based on the usage of computing resources. Static power consumption is a result of hardware design and is therefore of no use in power optimization efforts. This study reviews existing power measuring tools that are dynamic with the goal of evaluating and choosing the most appropriate.

3.5 Energy Modelling Tools

The conceptual framework identifies software defined power meters as the appropriate mechanism to measure the energy consumption of computing systems. A review of existing literature revealed a number of research studies that explore modern methodologies and develop software models that include JetsonLeap, EACOF, PEEK and PowerAPI

JetsonLeap

JetsonLeap (Bessa, Quint, Frank, & Quint, 2016) is complete power measurement infrastructure consisting of an NVIDIA Tegra TK1 board, power meter, simple electronic circuit and a C/C++ instrumentation library. Its design goals include measuring actual physical consumption, measure with greater granularity and work with a device that provides a general-purpose Input/Output such as BeagleBone, Arduino and Raspberry Pi.

This tool can be useful for this research as it is designed for System-on-a-Chip (SoC) devices such as Arduino and Raspberry Pi which will be used in this study's experiments. However, the compiler and hardware elements of the kit are a major drawback to the targeted high-level software of this study.

EACOF

EACOF (Field, Anderson, & Eder, 2014) uses a layer of abstraction between energy data sources and the applications. Through the use of two Application Programming Interfaces (API) developers can profile how their code consumes power. The two APIs include one that has access into the framework and another that provides access to the software application. The Provider-Consumer architecture of EACOF is simple and allows for portability.

PEEK

PEEK (Honig et al., 2014) is a holistic approach that covers both aspects of measurement and optimization. It combines both software and hardware measurements providing hints on where in the code to optimize for power consumption in an automated manner. Experiments using PEEK showed 8.4-fold speed-up of energy analysis and 25.3% improvements in energy consumption. Key design goals of PEEK are automation and adaptability. Consequently, no manual interventions are necessary.

3.6 PowerAPI: Software-Defined Power Meters

PowerAPI is a middleware toolkit that provides an architecture-agnostic way of measuring power consumed by software components. It achieves power measurement by allowing developers to create an assembly of power meters. It is an open source Application Programming Interface (API) licensed under AGPL v3 license. PowerAPI is implemented in Scala, utilising the Actor model and designed with modularity in mind.

PowerAPI works in two phases which are the Learning and Exploration steps. Learning step extracts Hardware Performance Counters (HPC) events from the CPU. The acquired HPC events are then ranked and regression analysis applied to them resulting in an analytical power model. Exploration follows and involves inputting the power model into the PowerAPI toolkit resulting in the power-defined software meters.

3.6.1 PowerAPI Architecture

PowerAPI is made up of separate power modules that are started and stopped at run-time. Sensor modules, which are dependent on the operating system, collect raw measurements from hardware.

Operating system-independent Formula modules in turn receive raw measurements from Sensor modules and use the developed Power Models to compute power consumption of each hardware component. In addition to metrics for hardware, Formula modules compute power consumption of processes and applications associated with the hardware component.

PowerAPI is automatically calibrated from a local database that contains configurations about hardware resources based on the environment. The Life Cycle module manages all modules and is responsible for adding, removing, starting, stopping, modifying modules based on user needs.

3.6.2 BitWatts: Process-level Power Models for Virtual Machines

BitWatts, (Colmant et al., 2015), is a multi-tier energy consumption approach that operates at the process level to calculate power consumption within virtual machines. Measurement of energy within a virtual machines is complex due to several factors that include directly accessing hardware, virtual machines that exceed the number of cores and having several applications running within a virtual machine. Communication across machines is achieved through a publish/subcribe communication channels aggregating energy metrics generated across multiple nodes.

Bitwatts implements the actor programming paradigm utilising three different actors that include Sensor, Formula and Reporter actors. Sensor actors collect coarse-grained raw measurements of the system. The raw measurements are drawn from from third-party power meters and internal probes or statistics from the procfs file system The Sensor actor then forwards the raw data to the Formula actor. Formula actors compute power consumption estimations using the raw data delivered by the sensor actor. The reporter actor then forwards the aggregated estimations to the Display object allowing the values to be accessible through either the web or the virtual file system. Display object's data can also be saved in a database. It also uses the libpfm4 library to increase the accuracy of the measurements.

3.6.3 WattsKit: Service-level Power Model

An energy consumption model utilising PowerAPI that measures energy within a distributed system was studied and implemented in (Colmant et al., 2015). WattsKit is a bottom-up service-level power that estimates power consumption of services spanning a number of hosts. The distributed power meter operates in phases. In the learning phase, it computes power consumption per node at the process level. In the aggregation phase WattsKit aggregates the power consumption of the targetted service across nodes within a cluster.

The WattsKit study case studied a deployment of the full-time search engine Elasticsearch on 1 master node, an Intel Xeon W3520 and 5 slave nodes, being 2 Intel Xeon W3520, 1 Intel Core2 Q6600 and 2 Intel Core2 E8400. Coordination and discovery services were implemented using Zookeper and deployment to nodes was implemented using Docker Swarm. The study accurately estimated the dynamic power consumption of nodes using HPCs as inputs. An update heavy workload of YCSB comprising of 50/50 reads and writes was executed on the search engine. The collected data was saved into a InfluxDB data store to allow for analysis.

An energy consumption model utilising PowerAPI that measures energy within a distributed system was studied and implemented in (Colmant et al., 2017). WattsKit is a bottom-up service-level power that estimates power consumption of services spanning a number of hosts. The distributed power meter operates in phases. In the learning phase it computes power consumption per node at the process level. In the aggregation phase WattsKit aggregates the power consumption of the targetted service across nodes within a cluster.

The WattsKit study case studied a deployment of the full-time search engine Elasticsearch on 1 master node, an Intel Xeon W3520 and 5 slave nodes, being 2 Intel Xeon W3520, 1 Intel Core2 Q6600 and 2 Intel Core2 E8400. Coordination and discovery services were implemented using Zookeper and deployment to nodes was implemented using Docker Swarm. An update heavy workload of YCSB comprising of 50/50 reads and writes was executed on the search engine. The collected data was saved into a InfluxDB data store to allow for analysis.

3.6.4 CodEnergy: Source-level Power Model

(Colmant, 2016) implemented an source code-level PowerAPI meter named CODENERGY. The modular CODENERGY system is composed of four components: codAgent, codEctor, codData and codVizu. codAgent collects raw data through statistical sampling and using sockets transmits the collected data to codEctor. codEctor in turn estimates the power consumption of the CPU and SSD using the values of the call stack saving the estimations into codData. codVizu can then be used to visualise the methods' power consumptions stored within codData.

The CODENERGY module was to analyse the energy consumption of two different versions, versions 2.2 and 3.2, of the Redis key-value, in-memory data structure server. In the experiment, Redis benchmark was used to 50 clients sending 100 000 commands each. Several methods within Redis including readQueryFromClient, sdscatlen and flushAppendOnlyFile were sampled for power consumption. The study concluded that energy consumption of sdscatlen of version 3.2 had improved over version version 2.2 whilst sdsnewlen deteriorated.

3.7 Cloudlet Architecture based on Raspberry Pi nodes

The feasibility of using the Raspberry Pi single board computer within cloudlet architectures has been on the rise. A review of literature of the use Raspberry Pi particularly in data intensive setups is thus follows.

The work of (Satyanarayanan et al., 2013) characterises certain contexts in which cloud computing services are inaccessible as hostile environments. It identifies three specific hostile environments that include natural disaster affected areas, developing countries with inadequate infrastructure and communities in war situations. The study encourages the development of a hierarchical extension of the cloud as opposed to viewing cloudlets as replacements of the traditional cloud. The proposal made is two have a two level architecture in which the cloud occupies level 1 whilst cloudlets make up level 2. The architecture leverages dynamic virtual machine synthesis through just-in-time provisioning of a cloudlet.

Experiments at Carnegie Mellon University using WiFi and 4G LTE (Gao et al., 2015) provided evidence that cloudlets improve performance by a factor of 51% whilst reducing the energy consumption of mobile devices by 42% . Through deliberately utilising a sub-standard cloudlet setup comprising of a 2.7Gz Dell computer, a 4GB against a more powerful EC2 instance the performance of the cloudlet was not expected to be very good. However, the results which were compared against an EC2-East instance showed improved response times as a result of offload location.

(Abrahamsson et al., 2013) developed a cluster of 300 Raspberry Pis as a testbed for green computing research as well as to explore use cases for cloudlets. The work also identifies strides in the use of Raspberry Pis within clusters at other location that includes University of Southampton (64 nodes), Boise State University (33) as well at the University of Glasgow (56 nodes). (Elkhatib et al., 2017) focused on establishing the feasibility of using micro-clouds to deliver fog computing services. The testbed used Raspberry Pis leveraging on the lightweight capabilities of Docker containers to isolate micro-cloud devices. Based on three concepts of hardware, resource management and programming abstractions the study confirmed the feasibility of cloudlets which were referred in the research as micro-clouds. With a modest budget of $9000 (Tso, White, Jouet, Singer, & Pezaros, 2013) developed an educational cloud environment using 200 Raspberry 2 Pi boards. Through the development of a metric to compute energy consumption of a hybrid Hadoop cluster, PerfEC comprising of traditional servers and 3 Raspberry Pi machines, (Anwar, Krish, & Butt, 2014) explored the feasibility of using single board computers as substrate Hadoop clusters. (Cloutier, Paradis, & Weaver, 2014) developed a Raspberry Pi cluster using seventeen different ARM development boards which allowed for per-node measurement of power consumption.

3.8 Times Series Data in IoT Applications

IoT applications support data intensive Cyber-Physical Systems that collect data at high velocity, volume and variety. Literature of how new time series focused databases have emerged and how traditional databases have been adapted to managed time series data is explored.

(Kopp & Bader, 2017) compares the performance of traditional SQL database management systems, NoSQL databases and Time Series Databases (TSDB). Using 10 different parameter that included four SCAN operation, 4 CRUD operations and SQL operators of AVG, SUM, COUNT, MAX and MIN the study created a framework highlighting how 25 different databases fared in supporting time series applications.

The study also explores how databases are utilising APIs such as HTTP/RESTful time series setups.

(Paethong et al., 2016) evaluates the performance of MySQL and MongoDB within an IoT middleware solution that has low energy footprint and supports the distribution of data on the x86 and ARM platforms. Results shows that both database performed faster on the x86 platform than on the Raspberry Pi for insertions.

3.9 Summary and Synthesis

The literature survey showed that an alternative to the cloud computing has merit when costs, privacy and latency are important factors. As a result, numerous researches are exploring the concept of cloudlets which do not rely on cloud computing. However, adopting cloudlets brings new challenges of data storage management in which cloud computing excels. Bringing data management on the edge of the resource-constrained local networks requires optimization of energy and performance. Literature reviewed also reveals how the indiffence of software developers exacerbates the challenge of optimising for power due to lack of solid and intuitive methodologies within tradiotional setups. Cloudlets, utilising non-traditional data storage mechanisms becomes is thus not a priority to developers. There is therefore a need to develop a suitable methodology and empirical evidence to encourage software developers to incorporate power optimisation during the development phase particularly in non-traditional IoT setups like cloudlets.

3.9.1 Balancing Power and Modern Features for Data management

Traditionally software vendors' primary performance indicator has been the rate of bringing out cutting edge technology in the shortest possible time-to-market. However, in a resource-constrained setup like cloudlets balancing between new features and energy savings is important. In the database management system domain, a number of features define the industry leaders. The most common features include:

- time-series workload management for IoT applications
- exposing database as restful end points,
- use of container technology to optimise devops

3.9.2 Generalisable Energy Models in Power Modelling

In summary, the related work literature reveals power measurements and optimizations efforts that require knowledge of the component, particularly the CPU, to be measured.

This can be attributed to the fact that most CPU manufacturers are building into the processor energy-saving features. This lack of platform-agnosticism when measuring power discourages software developers from measuring and optimising for power because of the level of hardware skills required. There is therefore a need to utilise power measurement methods that encourage software developers to make energy measurement and optimisation a priority alongside performance.

Because most software models rely on a deep understanding of the underlying system architecture to model power consumption. Developing generalizable energy models is a challenge in measuring energy consumption. This is a result of applications executing on different hardware, operating systems, environments and software domains (Colmant et al., 2017). There is therefore need to develop energy models that take into account the energy resulting from the differences in executing platforms before measuring the consumption attributable to the software.

4 Solution to the Problem

This section presents how the solution to the research problem was developed. The primary aim is develop a solution that is linear, intuitive and easy to understand. The solution utilised should allow for detailed comparisons of existing possibilities. The methodology must be reproducible and deterministic allowing users to infer, arrive at correct conclusions as well as predict outcomes. The study also aims to use the most independent approach that is free from the platform it is implemented on.

4.1 Approach

Numerous scientific approaches to achieving the study's research objectives exist. This study adopts an experimental approach to the objectives of the study in order to effectively meet the objectives of the study. A number of alternative experimental setups are available and these include simulations, emulations, profiling and testbeds.

- Objective 1: To determine if power consumption efficiencies are significant enough to warrant the resultant performance, functionality and security trade-offs.
    - Setting up a testbed will present data that can be compared between situations that prioritise functionality, in the form of web apis and performance, in the form of database optimisations, against energy consumption and the inverse
- Objective 2: To provide fine-grained energy consumption metrics for the cloudlet IoT architecture at the data storage layer using on hardware-agnostic tools.
    - To confirm whether measurements presented are agnostic of the platform on which they were collected its important to explore different platforms, that includes the native operating system against docker containerisation, in this study. Different testbed setups will allow for such in-depth comparisons of metrics from a wide array of platforms to confirm or dismiss the assertion.
- Objective 3: To develop a decision-making framework for IoT developers based energy efficiency of virtualised databases within cloudlet environments that utilise Single Board Computers as edge gateways.

  - This study targets IoT software developers, whom studies have shown, are sceptic of measuring power consumption of software due to lack of methodologies to test on their own. A testbed setup will allow developers to reproduce a setup and confirm the metrics provided.

Given the three research goals its apparent that testbeds offer the ideal metrics-based approach to solve the research problems.

4.2 Testbed Alternatives: Online and Offline Testbeds

There is a need to explore all possible testbed setups to arrive at the most optimal setup given research constraints. Testbeds can broadly be classified as online and offline testbeds. Online testbeds are available on the internet and enable one to focus on the research problem as the service providers will cater for setup requirements online. Online testbeds that were considered include FIT-IoTLAB, Industrial Internet Consortium, Fiesta-IoT and Eclipse IoT.

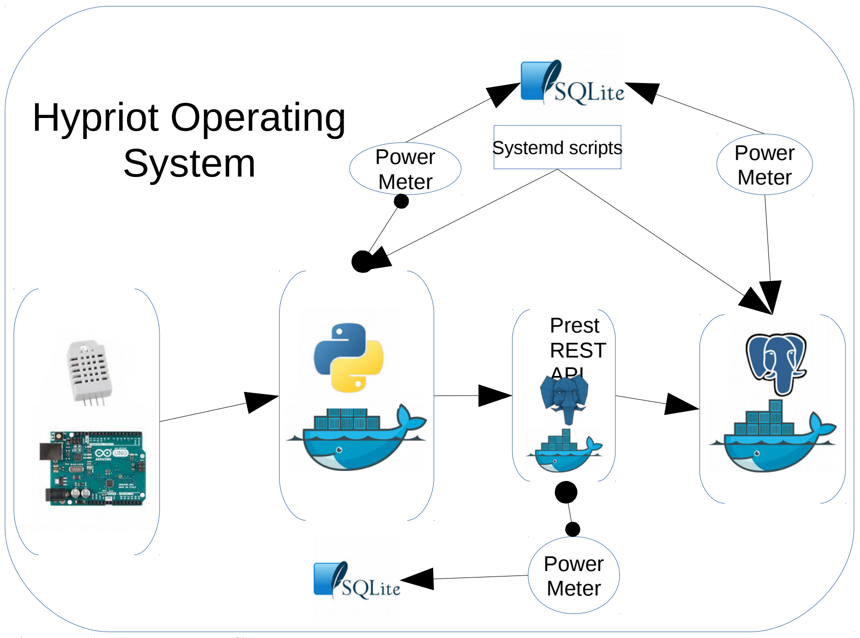
An offline testbed setup is one that is physically setup. Whilst online setups offers convenience and allow the researcher to focus on the research goals, this study chose to use offline testbed as it allows for more flexibility and the ability to easily adapt the environment iteratively.

The goals of the experiments to evaluate the energy efficiency of modern features databases under stress of streams of time series data. The database features under investigation for power consumption include:

- time-series data management
- exposing database as HTTP and RESTful end points,
- operation under Docker containerization

4.3 Materials

A suitable apparatus is required to ensure that the experiments meet the research objectives. The study intends to establish whether it is energy efficient to deploy database systems on a constrained gateway on a single board computer. Figure 1 below illustrates a possible testbed setup with all the apparatus required.



1. Figure 1: Apparatus Setup

4.3.1 DHT22 Sensors

DHT22 sensors will be used in collecting temperature and humidity values from the atmosphere. Table 1 below shows the specifications of the DHT22 temperature/humidity sensor.

	DHT22(AM2302)
Power Supply	3.3-5.5V DC
Output Signal	Digital vis 1-wire bus
Sensing Element	Polymer Humidity Capacitor
Operating Range	Humidity 0-100% temperature temperature -40~80°C
Accuracy	Humidity +-2%RH(Max +-5RH) temperature +-0.5°C
Resolution	Humidity 0.1%RH temperature +0.1 °
Repeatability	Humidity +-1%RH temperature +-0.2°

Table 1: DHT22 Specifications

4.3.2 Arduino Uno R3 Microcontroller

An Arduino Uno microcontroller will be connected to a temperature/humidity sensors, running an Arduino program that formats the collected values. Table 2 shows the specifications of the Arduino Uno used in the experiments.

	Arduino Uno R3
Microcontroller	Atmega328
Operating Voltage	5V
Input Voltage	7-12V
Digital I/O Pins	14 (6 with PWM)
Analog Input Pins	6
DC Current Per I/O Pin	40mA
DC Current for 3.3V	50mA
Flash Memory	bootloader
SRAM	2KB
EEPROM	1KB
Clock Speed	16MHz

Table 2: Arduino Uno R3 Specifications

4.3.3 Raspberry Pi 3 Single Board Computer

The Arduino board will be connected to a Raspberry Pi 3 computer using the serial connection. The single board computers will store the read temperature and humidity values into a database. A data sheet of the Raspberry Pi 3 single board computer that will be utilised in experiments is shown in Table 3.

	Raspberry Pi 3 Model B
Chipset	Broadcom BCM2837
CPU	Quad Core @1,2Gz ARM8 Cortex A-53
Memory	1GB LP-DDR2 900MHz
GPU	Broadcom VideoCore IV
Ethernet	10/100Mb/s
Flash Memory	MicroSD Card 32G
Connectity	USB 4xUSB 2.0 Host
OS	Raspbian

Table 3: Raspberry Pi 3 Specifications

The experiments will utilise three different types of databases. The DHT22 sensors will be in close proximity to ensure the temperature and humidity readings and the environment are the same. At any given time, the time series applications will be similar. The similar measurements will be maintained for a week before a new application is used with all the variables i.e. sensors, databases remaining the same. The time-series applications will be developed in Python.

4.3.4 Lightweight Operating Systems

Lightweight operating systems that will not significantly affect energy consumption will be installed on the Raspberry Pi. A trimmed down version of the official Raspbian operating system will deployed in some of the testbeds. Since the effect of Docker containers is central to this study a Docker optimised operating that is lightweight, Hypriot, will be used in some of the experiments as well.

4.3.5 Systemd Services and Timers

Each python application that was used to query the databases would have five systemd services associated with it. The first service is a timer service that is configured with the time when the service should start. The second service is responsible for starting two other services namely the meter service which spawns the PowerAPI power meter. Its also starts the save service which takes the metrics developed by the power meter saving them into an SQLite database for future analysis. The fourth service is a timer that is configured with the time the service should be stopped. The last service is the service that actually stops the service.

4.3.6 HTTP APIs

An HTTP API exposes a database to be queried transparently using any standard HTTP client such as curl. The study will test the energy efficiency of databases when processing queries through HTTP APIs. InfluxDB has built-in HTTP API support, thus additional API component when querying InfluxDB is not required. Prest will be used as the HTTP API interface of Postgres whilst Restheart will be used for MongoDB

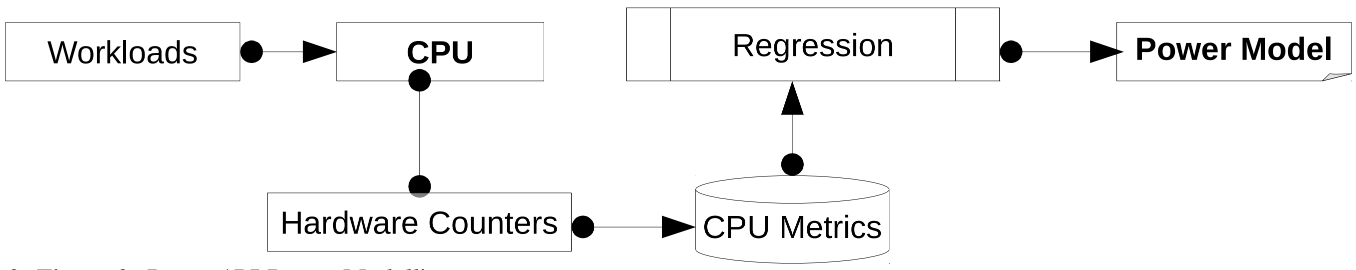
4.3.7 Generalisable Power Meters

Power meters will be developed from identifying the individual application and database processes and threads using systemd services. Scala objects also known as actors will be used to measure the power consumption. The Sensor actor will collect raw measurements from the underlying system forwarding them to the Formula Actor. The Formula Actor will in turn compute power estimation by implementing the power model. The Aggregator actor then aggregates power estimations using dimensions like process identifier or timestamp. Finally, the Reporter then formats estimations into a more readable way.

4.4 Energy Measurement and Saving Pipelines

The PowerAPI middleware software has to be configured to generate a software energy model that learns the CPU and using regression generates the power measurements. A general measurement pipeline that PowerAPI uses is shown in Figure 2.

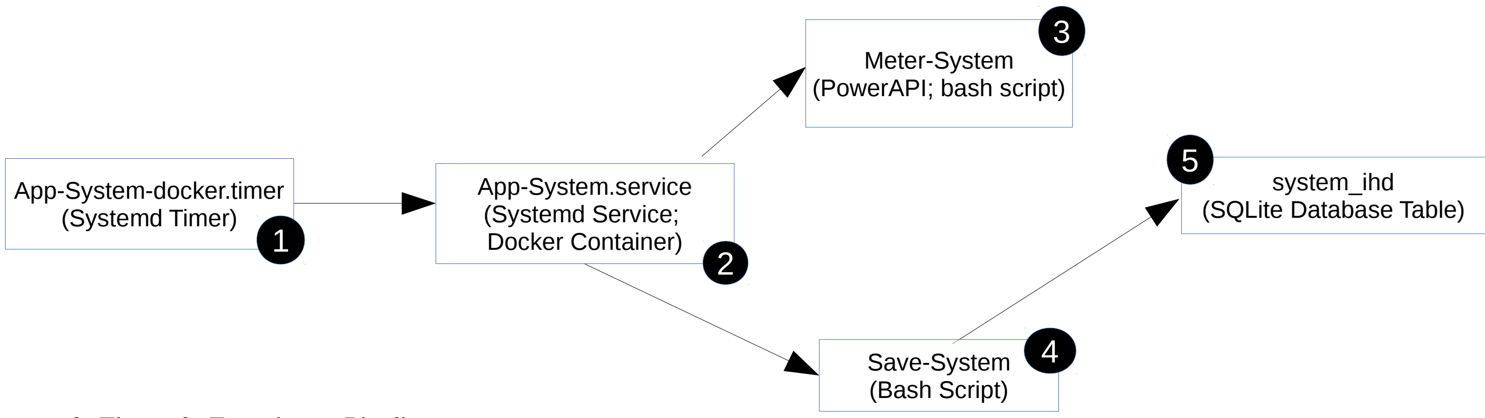
Several power meters are spawn for every process in the system; they include:



2. Figure 2: PowerAPI Power Modelling

1. The entire system (meter-system)
2. Python application that reads data from the Arduino microcontroller (meter-pypull-inserts, meter-pypush-inserts)
1. Python application that receives data from the database (meter-pypull-selects, meter-pypush-selects)
2. HTTP API that connects the python application to the database (meter-restheart, meter-prest)
3. The Database server (meter-influxdb, meter-mongodb, meter-postgres)

Scheduling the spawning of the processes is controlled by systemd timers. Figure 3 illustrates the steps that the scheduler follows as pipeline steps followed in a typical experiment:



3. Figure 3: Experiment Pipeline

The steps shown in Figure 3 above is summarised as follows:

1. The App-system-docker.timer is configured to start executing at a specific time.
2. When the specific time elapses the App-System service starts, and
3. Spawns a Docker container which and at the same time

4. Start a bash script that creates a PowerAPI power that starts computing the power consumption of the entire system and subsequently starts another bash that
5. Reads the energy metrics extracting the energy value and timestamp adds the name of the system event and saves the values in an SQLite table within a database.

5 Experiments

This section presents the experiments that were executed to measure the power consumption of different setups when saving data from the environment into time series databases. Three broadly similarly environments were setup as the testbeds whilst varying parameters of the ways of performing SQL insertions and selects on the databases. The database and application were deployed as Docker containers and utilised web APIs. as well as python database-specific APIs. to manipulate the data.

5.1 Execution Cycles

In order to correctly attribute energy to consumption to individual components of the system, a cycle of 6 runs was designed. The system software power meter runs in all the cycles in order to find correlations when the applications and databases are idle. The database would execute in five of the six runs. The runs are shown in the Table 4 below:

Cycle	Operation on Database	Opeartion API	Database	Time Interval	Power Meters Running
1	None	None	None	0000HRS-0259HRS	system
2	None	None	InfluxDB/Postgres/MongoDB	0300HRS-0559HRS	database, system
3	Insert	HTTP	InfluxDB/Postgres/MongoDB	0600HRS-0859HRS	application, database, system
4	Select	HTTP	InfluxDB/Postgres/MongoDB	0900HRS-1159HRS	application, database, system
5	Insert	Python	InfluxDB/Postgres/MongoDB	1200HRS-1459HRS	application, database, system
6	Select	Python	InfluxDB/Postgres/MongoDB	1500HRS-1759HRS	application, database, system

Table 4: Experimental Cycles

Each cycle would last 30 minutes. All the runs were implemented as systemd services.

The goal of the experiments is to apply the maximum feasible workload to the CPU through the database server and observe the energy consumption that can be directly attributed to the database server. In order to ensure that every database server used in the experiments worked in the most possible optimal state, all the three databases were optimised for performance beyond the default settings.

5.2 Systemd Services and Timers

Systemd services were created in the experiments to automate the starting and stopping of the database servers to ensure that the experiments are free from by human interventions

For a full list of all the systemd service and timers refer to Appendix XXIII.

5.3 Database Pre-experiment Optimisations

The default configurations of all the databases utilised in the experiments have unique characteristics that define their strengths. However, most of the unique features are not enabled by default for ease of installation and use. Some of the databases have platform-specific features that require adjustment to maximise performance on the execution environment.

5.3.1 Postgres Pre-Experiment Optimisations

In this environment, a Postgres-based RESTful API is utilised to manage interactions between the python application and the Postgres database. The application, the API and the database are implemented as Docker images that are spawn at the specific intervals by the systemd timers.

To isolate the CPU, which is the sole point of modelling for power, the study used a number of parameters that were modified before the experiments started. Huge pages were enabled to reduce the time spent on management memory. As the queries that are used to stress the CPU, thus the work_mem parameter was changed to 16MB. Shared buffers was set to 256MB to allow for increased workload on CPU. The keep-alive parameter was also set at 300 whilst file size, CPU time, virtual memory and memory size we set to unlimited. Since we had separate testbeds for each database the cacheSizeGB parameter was set 75% of the RAM thus 750MB.

5.3.2 MongoDB Pre-Experiment Optimisations

In the MongoDB experiment environment, a RESTful API is utilised to manage interactions between the python application and the database. The application, the API and the database are implemented as Docker images that are spawn at specific interval by the systemd timers.

As the study uses 32GB SD cards one parameter that can improve the performance of MongoDB is the dirty read background ratio which was to 5%. Since the queries in the study are not complex and the amount of data movement is not very voluminous a too high value for swappiness can compromise the CPU. Thus a swappiness value of 1 was used for the experiments.

For the native MongoDB instance, the deadline I/O scheduler was used for committing reads and writes whilst for the Docker version the noop scheduler was used. The read-ahead parameter which pre-fetches and caches requested blocks was set at 32 sectors.

5.3.3 InfluxDB Pre-Experiment Optimisations

InfluxDB has a built-in RESTful API implemented as the proprietary Line Protocol. In order to benefit from series cardinality the low-cardinality attributes of source and place were made tags whilst the volatile values of temperature, humidity and read_time were made fields. The pypull_insert python function which was also implemented as a Docker image commits data read from the Arduino microcontroller as batch points of size 500 utilising InfluxDB's Line Protocol. The stored data was downsampled through utilizing a retention policy of 21 days. Related data was placed in similar data. Though this is intuitive for a database experiment, this experiment consciously utilised this known caveat to ensure that the TSM engine compressed and made data available instantly. Since the Raspberry Pi is commodity hardware the experiments lowered the IOPS thresholds to 500. The experiments hosted a number of Docker containers to measure and save energy readings of the InfluxDB database in another database, SQLite3 and consequently, the experiments lowered the GOMAXPROCS environmental variable to 2 since the Raspberry Pi has 4 cores.

The focus of the study is establishing the power consumption of databases. However, in order to rule out any other contributing factors that affect power consumption, operating systems are varied to ascertain their effect on the databases. This is done to establish if there is a relationship between a database's energy consumption to the operating system the database is running on.

In order to attribute correctly the client process inserting data, the bash script that saves data of the energy consumption utilise array elements that correspond to the activated client at different time intervals. The five array elements have the following values:

- insert_API: for data inserted into the database through an HTTP API program
- insert_python: for data inserted into the database directly by a python program
- select_API: for data retrieved from the database by an HTTP API program
- select_python: for data retrieved from the database directly through a python program
- no_app: when the database is not receiving a requests from client applications

6 Results and Analysis

In this section, an analysis of the data generated by the power meters is conducted in order to facilitate inferences and discussion.

The results are presented as scenarios for convenient comparisons among the three databases. A total of twenty one (21) scenarios are presented, three for system experiments, twelve(12) for database experiments and six(6) for HTTP API clients experiments. In total nine(9) experiments were conducted to characterise and compare the entire system's energy consumption. Thirty-six (36) experiments on the three databases were done and twelve (12) experiments on HTTP API clients followed.

6.1 Units of Measure in PowerAPI Meters

In order to fully understand the power metrics derived from the power, the units of measure in relation to electronic devices need to be taken into account. The PowerAPI meters outputs power as milliwatts (mW) which is the equivalent of 0.0010 Joules/second.

6.2 Operating System Energy Consumption

Based on the experimental schedule cycle 1 has no application, API or Database execution. However, the data collected from the system power meter shows no variation in power consumption.

6.2.1 Native Minibian

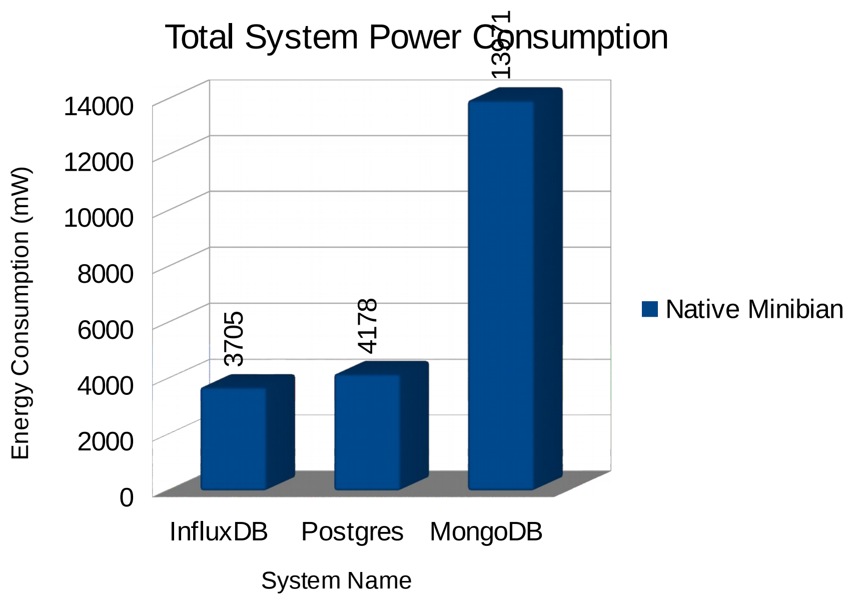


Table 5: System Power Consumption - Native Minibian

On native Minibian MongoDB consumed the most power of 13900mw with InfluxDB having the least consumption of 3705mW.

6.2.2 Minibian with Docker

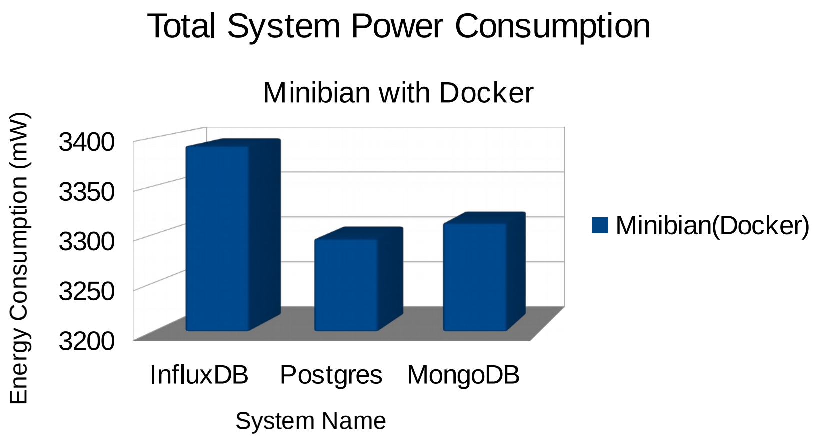


Table 6: Total Power Consumption Docker on Minibian

When implemented as Docker images all three databases have similar power consumption within the ranges of 3311mW, 3295 and 3392mW from Postgres, InfluxDB and MongoDB respectively.

6.2.3 Hypriot with Docker

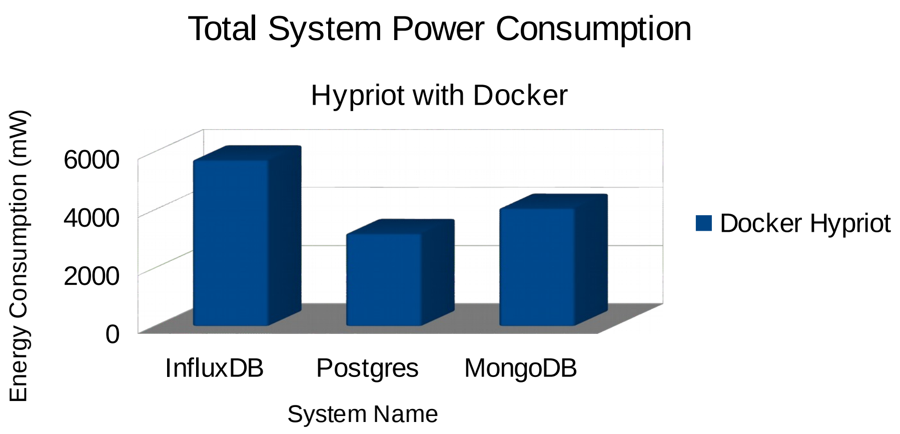


Table 7: System Power Consumption Hypriot with Docker

Postgres when packaged as a Docker container is very efficient consuming an average 3185mW whilst InfluxDB was inefficient consuming 5719mW.

6.2.4 Operating System Comparisons

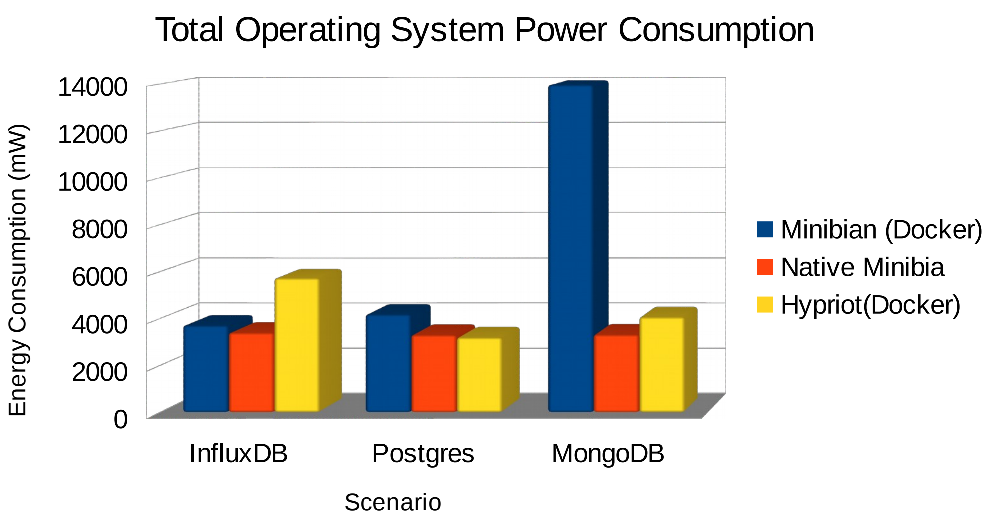


Table 8: System Power Consumption Comparisons

An analysis of the overall consumption of power consumption of the total consumption reveals a 45% increase when MongoDB is setup as a docker image on a Minibian operating system. All the other eight instances of the operating systems show a consistent total energy when no activity is taking place within the region of 3000 to 4000 mW of power.

6.3 Databases' Energy Consumption Comparisons

6.3.1 HTTP API Inserts

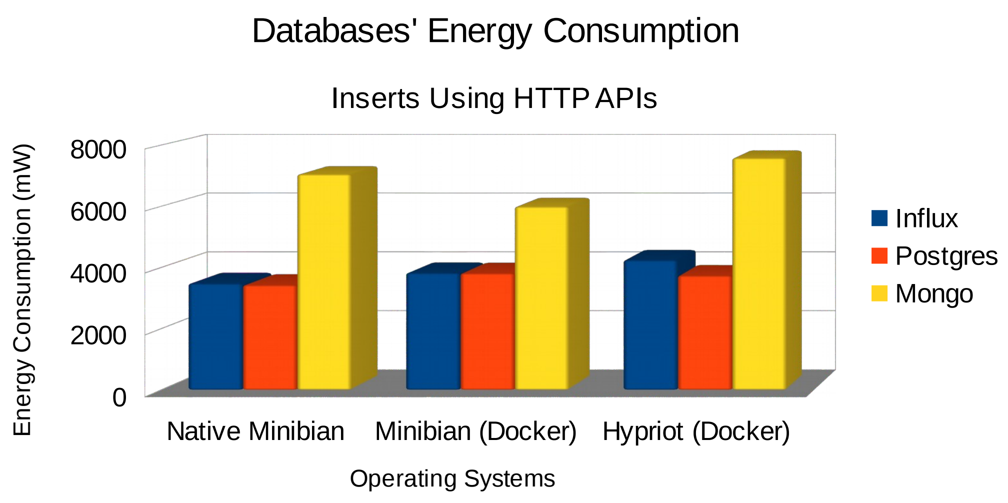


Table 9: Database Inserts using HTTP APIs

HTTP inserts on MongoDB consumed the most power on the Native Minibian as well as on the Hypriot operating systems on average of 6500mW.

6.3.2 Python API Inserts

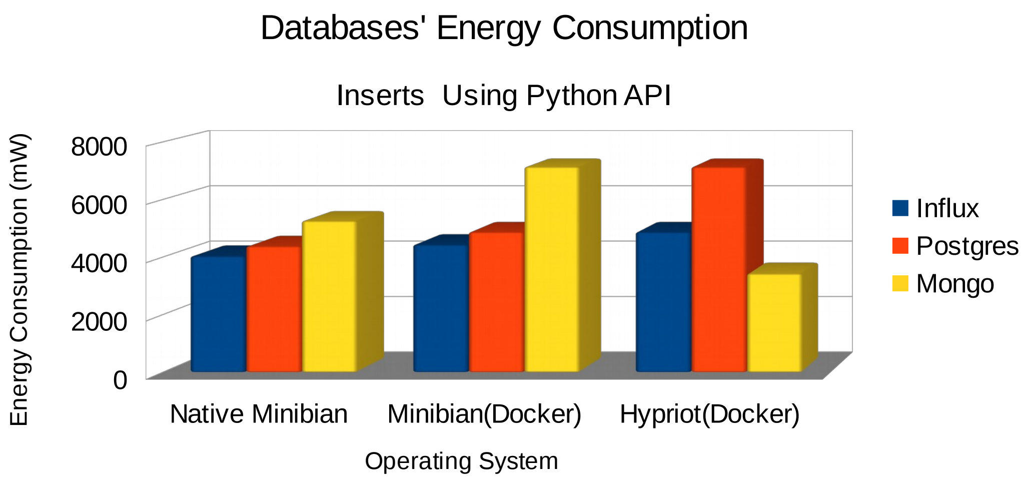


Table 10: Database Inserts using Python API

Inserting data was most expensive using the python API on native Minibian as well as on the virtualised Minibian when using MongoDB. The performance of Postgres and InfluxDB was identical. However, Postgres was inefficient on Hypriot.

HTTP APIs Selects

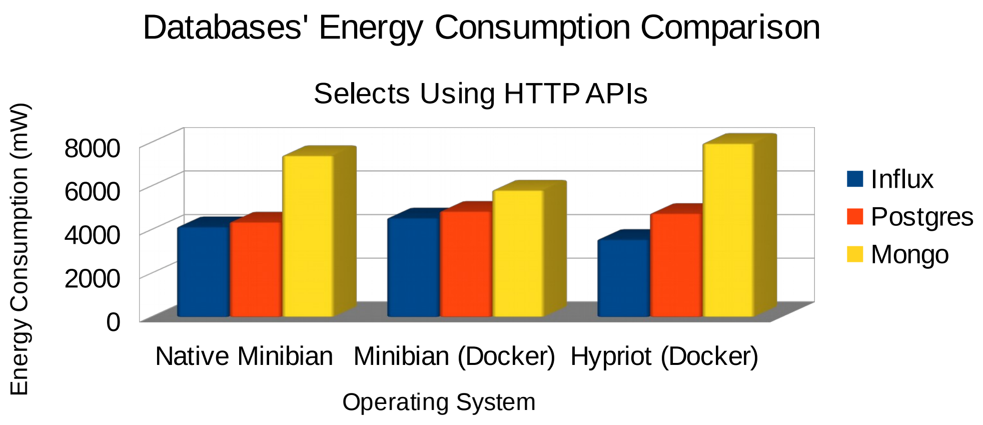


Table 11: Database Selects using HTTP APIs

Selecting data was most expensive using the python API on all platforms when using MongoDB. The performance of Postgres and InfluxDB was identical.

Python API Selects

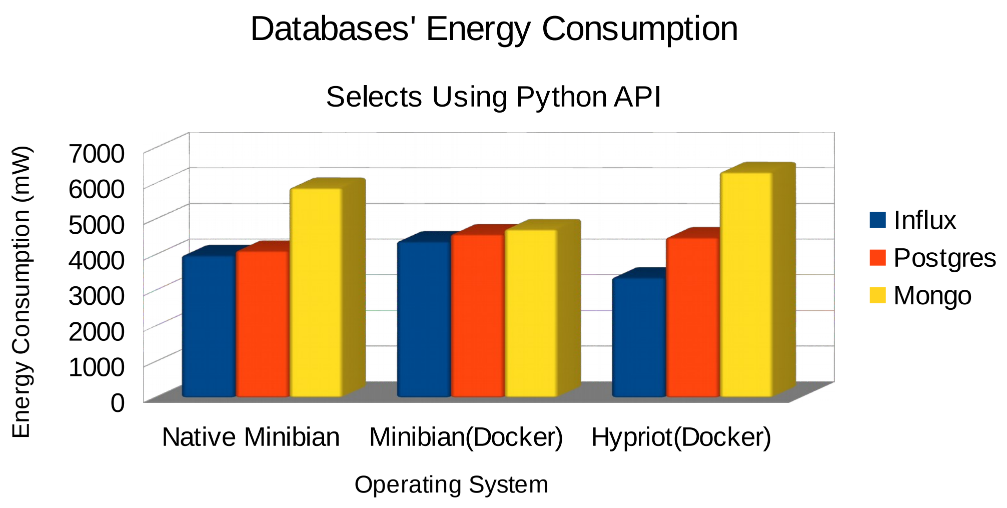


Table 12: Database Selects using Python APIs

7 Discussion

In this section, an interpretation of the relationships among the data is made. This inference from the data is meant to answer the research questions arising from the research objectives that were identified at the inception of the study, namely:

- Are the energy savings significant enough to warrant power optimisation in cloudlets?

With the least overall system value of around 3300mW and a peak of 13000mW, it is apparent that there are significant difference with regards the energy consumption of database servers.

- Which database(s) is/are energy efficient when handling novel features that include HTTP APIs, Docker containerisation and implementation of time series data management?

MongoDB on all the three platforms had the least efficiency. Postgres on the native operating system as well on virtualised on Minibian had the best figures in terms of energy efficiency. InfluxDB deployed as docker images on Hypriot was the most efficient.

The fact that on average 45% increase when docker containers were used suggests that most all the three database servers are not production ready on Raspberry Pi.

The Prest HTTP API used with MongoDB was very energy efficient than the Restheart API used with MongoDB. Prest's consumption as Docker containers was lowest at 1464mW on Hypriot when selecting data compared to 3377mW consumption by Restheart. When inserting data Prest had the lowest consumption as Docker container of 1257mW on Hypriot whilst RestHeart had a value and highest values of 1543mW on Native Minibian.

Experiments on InfluxDB, MongoDB and Postgres conducted on the Raspberry Pi 3 were aimed at establishing the most energy-efficient databases considering parameters that including use as Docker containers as well as manipulation of data using HTTP APIs.

In native environments, InfluxDB was the most energy efficient database server. However, Postgres was the most efficient when packaged as a Docker image. When data was manipulated using HTTP APIs, the Prest HTTP API that manipulates Postgres was the most efficient.

As a decision making framework this study concludes that using HTTP API to manipulate data does not result in significant power consumption than directly using the Python API. Postgres using an external HTTP API in Prest and InfluxDB utilising its internal REST API, Line Protocol, consumed way less power than when data was manipulated using direct Python APIs. The slight differences between native instances of both the database and the HTTP APIs and virtualised instances in Docker point to the fact that Docker virtualization did not result in significant power consumption.

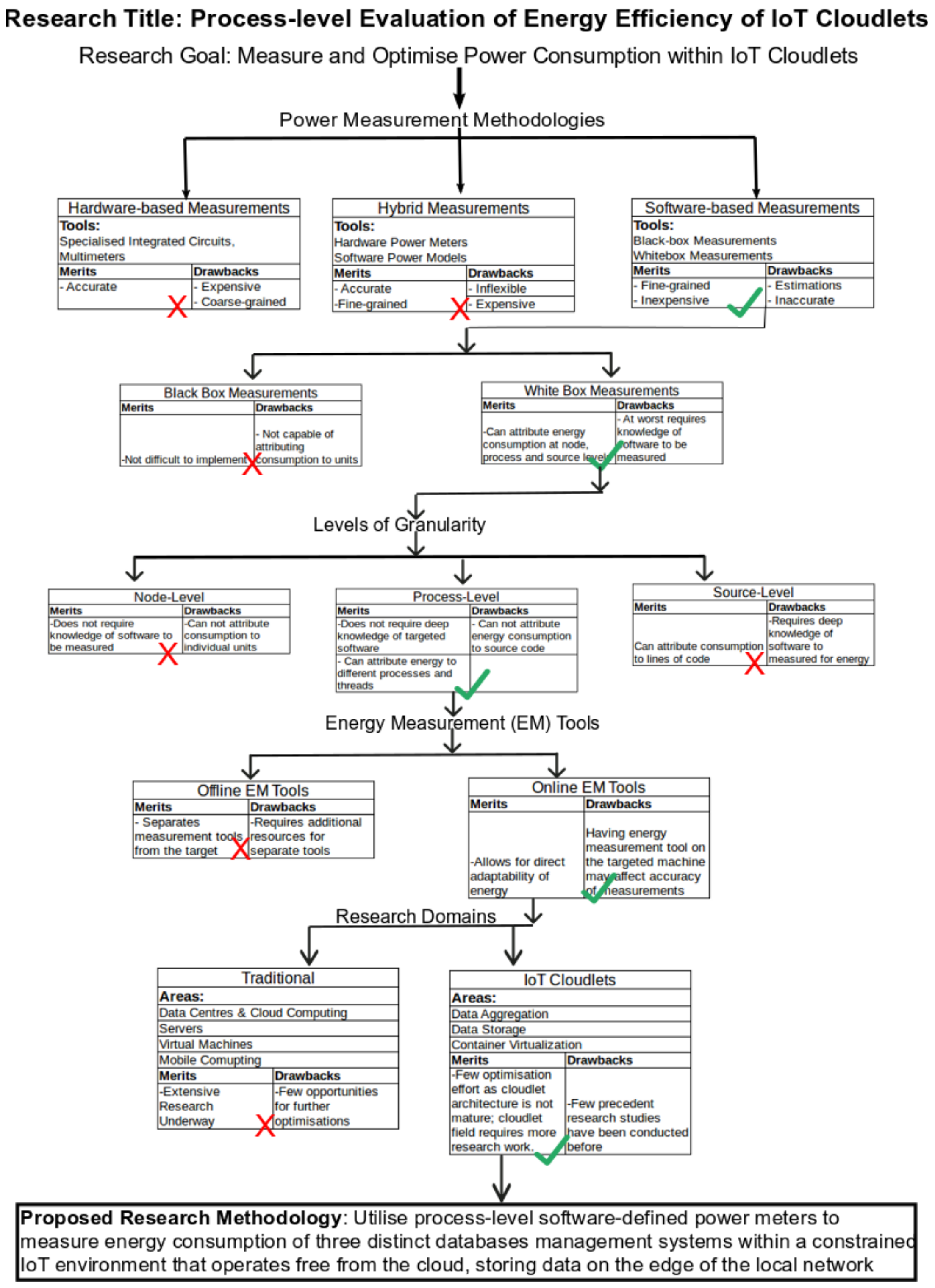
8 Conclusion

In conclusion, the Raspberry Pi has the ability to host database applications that support basic time series applications without incurring excessive energy costs. Based on the results of the experiments Postgres and InfluxDB are energy efficient when deployed on Raspberry Pi. The study also revealed that interacting with databases using web APIs does not have significant energy costs. In summary, the consistent results of the experiments show that PowerAPI can be used as a generalisable power modelling as it learned the ARM architecture and developed power meters on-the-fly.

There is room for future work through extending the single Raspberry Pi gateway into a cluster that is managed by an open source cloud computing platform ideally deployed using tools such as OpenStack.

9 Appendices

Appendix I: Conceptual Framework


Research Title: Process-level Evaluation of Energy Efficiency of IoT Cloudlets
Research Goal: Measure and Optimise Power Consumption within IoT Cloudlets
Power Measurement Methodologies
Hardware-based Measurements
Tools:
Specialised Integrated Circuits, Multimeters
Merits
Drawbacks
- Accurate
- Expensive
- Coarse-grained
Hybrid Measurements
Tools:
Hardware Power Meters
Software Power Models
Merits
Drawbacks
- Accurate
-Fine-grained
- Inflexible
- Expensive
Software-based Measurements
Tools:
Black-box Measurements
Whitebox Measurements
Merits
Drawbacks
- Fine-grained
- Inexpensive
- Estimations
- Inaccurate
Black Box Measurements
Merits
Drawbacks
-Not difficult to implement
- Not capable of attributing consumption to units
White Box Measurements
Merits
Drawbacks
-Can attribute energy consumption at node, process and source level
- At worst requires knowledge of software to be measured
Levels of Granularity
Node-Level
Merits
Drawbacks
-Does not require knowledge of software to be measured
-Can not attribute consumption to individual units
Process-Level
Merits
Drawbacks
-Does not require deep knowledge of targeted software
- Can attribute energy to different processes and threads
- Can not attribute energy consumption to source code
Source-Level
Merits
Drawbacks
Can attribute consumption to lines of code
-Requires deep knowledge of software to measured for energy
Energy Measurement (EM) Tools
Offline EM Tools
Merits
Drawbacks
- Separates measurement tools from the target
-Requires additional resources for separate tools
Online EM Tools
Merits
Drawbacks
-Allows for direct adaptability of energy
Having energy measurement tool on the targeted machine may affect accuracy of measurements
Research Domains
Traditional
Areas:
Data Centres & Cloud Computing
Servers
Virtual Machines
Mobile Comupting
Merits
Drawbacks
-Extensive Research Underway
-Few opportunities for further optimisations
IoT Cloudlets
Areas:
Data Aggregation
Data Storage
Container Virtualization
Merits
Drawbacks
-Few optimisation effort as cloudlet architecture is not mature; cloudlet field requires more research work.
-Few precedent research studies have been conducted before
Proposed Research Methodology: Utilise process-level software-defined power meters to measure energy consumption of three distinct databases management systems within a constrained IoT environment that operates free from the cloud, storing data on the edge of the local network

Appendix II: Scenario 1- System's Energy Metrics on Native Minibian

	Native Minibian		
	InfluxDB	**Postgres**	**MongoDB**
Average Power Consumption (mW)	3705.0706081692	4177.8054427785	13970.7640768953

Appendix III: Scenario 2 – System's Energy Metrics on Virtualised Minibian

	Virtualised Minibian		
	InfluxDB	**Postgres**	**MongoDB**
Average Power Consumption (mW)	3391.36828682646	3295.58808091447	3311.88828368587

Appendix IV: Scenario 3 – System's Energy Metrics on Virtualised Hypriot

	Virtualised Hypriot		
	InfluxDB	**Postgres**	**MongoDB**
Average Power Consumption(mW)	5719.51611755621	3185.86272248836	4057.7817177488

Appendix V: Scenario 4 – HTTP Inserts on Native Minibian

	HTTP Inserts on Native Minibian		
	InfluxDB	Postgres	MongoDB
Average Energy Consumption (mW)	3477.6043	3425.0981	7048.4716

Appendix VI: Scenario 5: Python Inserts on Native Minibian

	Python Inserts on Native Minibian		
	InfluxDB	Postgres	MongoDB
Average Energy Consumption (mW)	4035.9473	4380.1925	5258.1058

Appendix VII: Scenario 6 – HTTP Selects on Native Minibian

	HTTP Selects on Native Minibian		
	InfluxDB	**Postgres**	**MongoDB**
Average Energy Consumption (mW)	4145.1304	4365.471	7410.7343

Appendix VIII: Scenario 7 – Python Selects on Native Minibian

	Python Selects on Native Minibian		
	InfluxDB	**Postgres**	**MongoDB**
Average Energy Consumption (mW)	3985.5974	4116.1306	5879.9152

Appendix IX: Scenario 8: HTTP Inserts on Virtualised Minibian

HTTP Selects on Virtualised Minibian			
	InfluxDB	**Postgres**	**MongoDB**
Average Energy Consumption (mW)	4555.0883	4855.9188	5824.1275

Appendix X: Scenario 9 – Python Inserts on Virtualised Minibian

Python Inserts on Virtualised Minibian			
	InfluxDB	Postgres	MongoDB
Average Energy Consumption (mW)	4435.1069	4872.2943	7135.2487

Appendix XI: Scenario 10 – HTTP Selects on Virtualised Minibian

HTTP Selects on Virtualised Minibian			
	InfluxDB	**Postgres**	**MongoDB**
Average Energy Consumption (mW)	4555.0883	4855.9188	5824.1275

Appendix XII: Scenario 11 – Python Selects on Virtualised Minibian

Python Selects on Virtualised Hypriot			
	InfluxDB	**Postgres**	**MongoDB**
Average Energy Consumption (mW)	3366.5925	4488.79	6322.4895

Appendix XIII: Scenario 12 – HTTP Inserts on Virtualised Hypriot

HTTP Inserts on Virtualised Hypriot			
	InfluxDB	Postgres	MongoDB
Average Energy Consumption (mW)	4241.5208	3734.5639	7579.0017

Appendix XIV: Scenario 13 – Python Inserts on Virtualised Hypriot

Python Inserts on Virtualised Hypriot			
	InfluxDB	Postgres	MongoDB
Average Energy Consumption (mW)	4872.2943	7135.2487	3425.0981

Appendix XV: Scenario 14 – HTTP Selects on Virtualised Hypriot

HTTP Selects on Virtualised Hypriot			
	InfluxDB	**Postgres**	**MongoDB**
Average Energy Consumption (mW)	3570.5285	4760.7047	7968.5315

Appendix XVI: Scenario 15 – Python Selects on Virtualised Hypriot

Python Selects on Virtualised Hypriot			
	InfluxDB	**Postgres**	**MongoDB**
Average Energy Consumption (mW)	3366.5925	4488.79	6322.4895

Appendix XVII: Scenario 16 – HTTP API Clients' Inserts on Native Minibian

HTTP Inserts on Native Minibian		
	Prest(Postgres)	RestHeart(MongoDB)
Average Energy Consumption (mW)	1543.34546288356	4123.4174495717

Appendix XVIII: Scenario 17 – HTTP API Clients' Inserts on Virtualised Minibian

HTTP Inserts on Virtualised Minibian		
	Prest(Postgres)	RestHeart(MongoDB)
Average Energy Consumption (mW)	1357.75292598136	3473.76740579804

Appendix XIX: Scenario 18 – HTTP API Clients' Inserts on Virtualised Hypriot

HTTP Inserts on Virtualised Hypriot		
	Prest(Postgres)	RestHeart(MongoDB)
Average Energy Consumption (mW)	1257.07802645799	4808.07076366705

Appendix XX: Scenario 19 – HTTP API Clients' Selects on Native Minibian

HTTP Selects on Native Minibian		
	Prest(Postgres)	**RestHeart(MongoDB)**
Average Energy Consumption (mW)	1480.18110035083	3909.38311701013

Appendix XXI: Scenario 20 – HTTP API Clients' Selects on Virtualised Minibian

HTTP Selects on Virtualised Minibian		
	Prest(Postgres)	**RestHeart(MongoDB)**
Average Energy Consumption (mW)	1780.35239323832	3377.99395718569

Appendix XXII: Scenario 21 – HTTP API Clients' Selects on Virtualised Hypriot

HTTP Selects on Virtualised Hypriot		
	Prest(Postgres)	**RestHeart(MongoDB)**
Average Energy Consumption (mW)	1464.08971294003	5015.40257426082

Appendix XXIII: Systemd Service and Timers

Name	**Description**
App-influxdb-docker.service	The service that is started by the timer which in turn starts the InfluxDB docker container, InfluxDB power meter and the save module which saves energy metrics into SQLite Database
App-influxdb-docker-stop.service	The service that stops the InfluxDB container, meter and saving module service as specified by the app-influential service
App-influxdb-docker-stop.timer	A timer that holds the timer to stop the InfluxDB container, meter and saving module.
App-influxdb-docker.timer	A timer that holds the timer to start the InfluxDB container, meter and saving module.
App-pypull-insert-docker.service	The service that is started by the timer which in turn starts the pypull-insert docker container, pypull-insert power meter and the save module which saves energy metrics into SQLite Database
App-pypull-insert-docker-stop.service	The service that stops the pypull-insert container, meter and saving module service as specified by the app-pypull-insert.timer service
App-pypull-insert-docker-stop.timer	A timer that holds the timer to stop the pypull-insert container, meter and saving module.
App-pypull-insert-docker.timer	A timer that holds the timer to start the pypull-insert container, meter and saving module.
App-pypull-select-docker.service	The service that is started by the timer which in turn starts the pypull-select docker container, pypull-select power meter and the save module which saves energy metrics into SQLite Database
App-pypull-select-docker-stop.service	The service that stops the pypull-select container, meter and saving module service as specified by the app-pypull-select.timer service
App-pypull-select-docker-stop.timer	A timer that holds the timer to stop the pypull-select container, meter and saving module.
App-pypull-select-docker.timer	A timer that holds the timer to start the pypull-select container, meter and saving module.
App-pypush-insert-docker.service	The service that is started by the timer which in turn starts the pypush-insert docker container, pypush-insert power meter and the save module which saves energy metrics into SQLite Database
App-pypush-insert-docker-stop.service	The service that stops the pypush-insert container, meter and saving module service as specified by the app-pypush.timer service
App-pypush-insert-docker-stop.timer	A timer that holds the timer to stop the pypush-insert container, meter and saving module.

App-pypush-insert-docker.timer	A timer that holds the timer to start the pypull-insert container, meter and saving module.
App-pypush-select-docker.service	The service that is started by the timer which in turn starts the pypush-insert docker container, pypush-insert power meter and the save module which saves energy metrics into SQLite Database
App-pypush-select-docker-stop.service	The service that stops the pypush-select container, meter and saving module service as specified by the app-pypush.timer service
App-pypush-select-docker-stop.timer	A timer that holds the timer to stop the pypush-select container, meter and saving module.
App-pypush-select-docker.timer	A timer that holds the timer to start the pypush-select container, meter and saving module.
Influxdb-docker.service	The service that spawns the InfluxDB image into a container
Pypull-insert-to-influxdb.service	The service that spawns the pypull-insert-to-influxdb image into a container
Pypull-insert-docker.service	The service that spawns the pypull-insert image into a container
Pypull-insert-docker.service	The service that spawns the pypull-insert image into a container
Pypush-insert-docker.service	The service that spawns the pypush-insert image into a container
Pypush-select-docker.service	The service that spawns the pypush-select image into a container
Meter-system.service	Spawns the PowerAPI power meter for the whole system
Meter-influxdbdocker.service	Spawns the PowerAPI power meter for InfluxDB container
Meter-Pypull-insert-docker.service	Spawns the PowerAPI power meter for pypull-insert container
Meter-Pypush-insert-docker.service	Spawns the PowerAPI power meter for pypull-select container
Meter-Pypush-insert-docker.service	Spawns the PowerAPI power meter for pypush-insert container
Meter-Pypush-select-docker.service	Spawns the PowerAPI power meter for pypush-select container
Save-system.service	Save energy metrics of the whole system
Save-influxdb.service	Saves energy metrics of the InfluxDB service
Save-Pull-insert-docker.service	Saves energy metrics of the pypull-insert service
Save-Pull-select-docker.service	Saves energy metrics of the pypull-select service
Save-Pushy-insert-docker.service	Saves energy metrics of the pypush-insert service
Save-Pypush-select-docker.service	Saves energy metrics of the pypush-select service

10 References

Abrahamsson, P., Helmer, S., Oyetoyan, T. D., Cardano, F., Gadler, D., Morandini, D., Mahabub, A. (2016). Bringing the Cloud to Rural and Remote Areas – Cloudlet by Cloudlet, https://arxiv.org/abs/1605.03622. 1–9.

Abrahamsson, P., Helmer, S., Phaphoom, N., Nicolodi, L., Preda, N., Miori, L., … Bugoloni, S. (2013). Affordable and Energy-Efficient Cloud Computing Clusters : The Bolzano Raspberry Pi Cloud Cluster Experiment. 2013 IEEE 5th International Conference on Cloud Computing Technology and Science, Bristol, 2013, pp. 170-175. doi: 10.1109/CloudCom.2013.121

Acar, H., Alptekin, G. I., Gelas, J., Ghodous, P., Acar, H., Alptekin, G. I., … Ghodous, P. (2016). Beyond CPU : Considering Memory Power Consumption of Software To cite this version : HAL Id : hal-01314070

Al-Rousan, M., Al-Shara, E., Jararweh, Y., & Alshayeji, M. H. (2017). Cloudlet-Based ad hoc mobile cloud Computing: Design and evaluation. *International Journal of Embedded Systems*, *9*(5). https://doi.org/10.1504/IJES.2017.086727

Anwar, A., Krish, K. R., & Butt, A. R. (2014). On the Use of Microservers in Supporting Hadoop Applications.

Colmant, M. (2016). Multi-Dimensional Analysis of Software Power Multi-Dimensional Analysis of Software Power Consumptions in Multi-Core Architectures THÈSE.

B, Y. W., He, Y., Cao, T., & Kondo, M. (2018). *A Power Management Framework with Simple DSL for Automatic on Power-Constrained HPC Systems* (Vol. 1). Springer International Publishing. https://doi.org/10.1007/978-3-319-69953-0

Babu, A. (2014). Power Optimization Techniques at Circuit and Device Level in Digital CMOS VLSI – A Review, *3*(11), 375–379.

Bessa, T., Quint, P., Frank, M., & Quint, F. M. (2016). JetsonLeap : A Framework to Measure Energy-Aware Code Optimizations in Embedded and Heterogeneous Systems, 16–30. https://doi.org/10.1007/978-3-319-45279-1

Bridges, R. A., Imam, N., & Mintz, T. M. (2016). Understanding GPU Power: A Survey of Profiling, Modeling, and Simulation Methods. *ACM Computing Surveys*, *49*(3), 1–27. https://doi.org/10.1145/2962131

Cloutier, M. F., Paradis, C., & Weaver, V. M. (2014). Design and Analysis of a 32-bit Embedded High-Performance Cluster Optimized for Energy and Performance. 2014 Hardware-Software Co-Design for High Performance Computing, New Orleans, LA, 2014, pp. 1-8. doi: 10.1109/Co-HPC.2014.7

Colmant, M., Felber, P., Rouvoy, R., Seinturier, L., Colmant, M., Felber, P., Seinturier, L. (2017). WATTSKIT : Software-Defined Power Monitoring of Distributed Systems. *17th IEEE/ACM International Symposium on Cluster, Cloud and Grid*

Computing (CCGRID), Madrid, 2017, pp. 514-523. doi: 10.1109/CCGRID.2017.27

Colmant, M., Kurpicz, M., Felber, P., Sobe, A., Colmant, M., Kurpicz, M., Sobe, A. (2015). Process-level Power Estimation in VM-based Systems. EuroSys '15 Proceedings of the Tenth European Conference on Computer Systems Article No. 14

Combarro, M., Tchernykh, A., Kliazovich, D., Drozdov, A., & Radchenko, G. (2016). Energy-Aware Scheduling with Computing and Data Consolidation Balance in 3-Tier Data Center. In *2016 International Conference on Engineering and Telecommunication (EnT)* (pp. 29–33). https://doi.org/10.1109/EnT.2016.015

Constant, N., Borthakur, D., Abtahi, M., Dubey, H., & Mankodiya, K. (2017). Fog-Assisted wIoT : A Smart Fog Gateway for End-to-End Analytics in Wearable Internet of Things, 23rd IEEE Symposium on High Performance Computer Architecture. 1–5.

Devon M. Simmonds. (2012). The Programming Paradigm Evolution. *Computer*, *45*, 93–95. https://doi.org/doi.ieeecomputersociety.org/10.1109/MC.2012.219

Dong, M., & Zhong, L. (2011). Self-Constructive High-Rate System Energy Modeling for Battery-Powered Mobile Systems. MobiSys '11 Proceedings of the 9th international conference on Mobile systems, applications, and services. 335-348

Dongarra, J., Ltaief, H., Luszczek, P., & Weaver, V. M. (2012). Energy Footprint of Advanced Dense Numerical Linear Algebra using Tile Algorithms on Multicore Architecture. *Second International Conference on Cloud and Green Computing*, Xiangtan, 2012, pp. 274-281. doi: 10.1109/CGC.2012.113

Elkhatib, Y., Porter, B., Ribeiro, H. B., Zhani, M. F., Qadir, J., & Riviere, E. (2017). On using micro-clouds to deliver the fog. *IEEE Internet Computing*, *21*(2), 8–15. https://doi.org/10.1109/MIC.2017.35

Honig, T., Janker, H., Eibel, C., Schroder-Preischat, W., Mihelic, O., Kapitza, R. (2014). Proactive energy-aware programming with PEEK, TRIOS'14 Proceedings of the 2014 International Conference on Timely Results in Operating Systems,1–6.

Field, H., Anderson, G., & Eder, K. (2014). EACOF : A Framework for Providing Energy Transparency to enable Energy-Aware Software Development, Proceedings of the 29th Annual ACM Symposium on Applied Computing, 1–7.

Fredriksson, T., Barayre, C., Shubhangi Denblyden, S. F. G., Jang, S., Korka, D., Lakhe, S., & Sicat, M. (2013). *Information Economy report 2013: The Cloud Economy and Developing Countries*. *United Nations*. Retrieved from http://unctad.org/en/pages/PublicationWebflyer.aspx?publicationid=710

Gallo, S. L., Chichizola, F., & Giusti, L. De. (2018). Analysis of RAPL Energy Prediction Accuracy in a Matrix Multiplication Application on Shared Memory Analysis of RAPL Energy Prediction Accuracy in a Matrix Multiplication

Application on Shared Memory, (January). https://doi.org/10.1007/978-3-319-75214-3

Gao, Y., Hu, W., Ha, K., Amos, B., Pillai, P., & Satyanarayanan, M. (2015). Are Cloudlets Necessary?, Intel Labs, (October).

Hao, S., Li, D., Halfond, W. G. J., & Govindan, R. (2013). Estimating Mobile Application Energy Consumption using Program Analysis, *35th International Conference on Software Engineering (ICSE)*, San Francisco, CA, 2013, pp. 92-101. doi: 10.1109/ICSE.2013.6606555

Hindle, A. (2016). Green software engineering : the curse of methodology Green Software Engineering : The Curse of Methodology. *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Suita, 2016, pp. 46-55.
doi: 10.1109/SANER.2016.60

Hosseini, M., Peters, J., & Shirmohammadi, S. (2013). Energy-Budget-Compliant Adaptive 3D Texture Streaming in Mobile Games. *Proc of MMSys*, (March 2015), 1–11. https://doi.org/10.1145/2483977.2483978

Jeyanthi, N., Thandeeswaran, R., & IGI Global. (2017). Security breaches and threat prevention in the internet of things, (March), 52. https://doi.org/10.4018/978-1-5225-2296-6

Johnston, S. J., & Cox, S. J. (2017). The Raspberry Pi : A Technology Disrupter , and the Enabler of Dreams. https://doi.org/10.3390/electronics6030051

Kim, Y. G., Kim, M., Kim, J. M., Sung, M., & Chung, S. W. (2015). A novel GPU power model for accurate smartphone power breakdown. *ETRI Journal*, *37*(1), 157–164. https://doi.org/10.4218/etrij.14.0113.1411

Koomey, J. G., Berard, S., Sanchez, M., & Wong, H. (2011). Implications of Historical Trends in the Electrical Efficiency of Computing, 46–54.

Kopp, O., & Bader, A. (2017). Survey and Comparison of Open Source Time Series Databases, Workshopband, Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn, 249–268.

Kortas, N., Azzabi, H., Arbia, A. Ben, & Tahar, J. B. (2015). Communication within Cloudlet using the Raspberry. *Procedia Computer Science*, *73*(Awict), 193–198. https://doi.org/10.1016/j.procs.2015.12.012

Lago, P. (2015). Challenges and Opportunities for Sustainable Software, 10–11. https://doi.org/10.1109/PLEASE.2015.8

Lim, J., & Lakshminarayana, N. (2013). Power Modeling for GPU Architecture using McPAT. Georgia Institute, 1–17. ACM Transactions on Design Automation of Electronic Systems (TODAES) Volume 19 Issue 3, June 2014, Article No. 26

Manotas, I., Bird, C., Clause, J. A., Manotas, I., Bird, C., Zhang, R., & Shepherd, D. (2016). An empirical study of practitioners ' perspectives on green software engineering An Empirical Study of Practitioners ' Perspectives on Green Software Engineering, (May). https://doi.org/10.1145/2884781.2884810

Marcus, H., Marcus, V., & Hermann, H. (2012). Measuring Energy Consumption for Short Code Paths Using RAPL. ACM SIGMETRICS Performance Evaluation Review 40(3) DOI: 10.1145/2425248.2425252.

Morabito, R. (2017). Virtualization on internet of things edge devices with container technologies: A performance evaluation. *IEEE Access*, *5*, 8835–8850. https://doi.org/10.1109/ACCESS.2017.2704444

Morabito, R., & Benchmark, A. S. (2016). A Performance Evaluation of Container Technologies on Internet of Things Devices. *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WORKSHOPS)*, San Francisco, CA, 2016, pp. 999-1000. doi: 10.1109/INFCOMW.2016.7562228

Mthunzi, S. N., Benkhelifa, E., Jararweh, Y., & Al-Ayyoub, M. (2017). Cloudlet solution for digital forensic investigation of multiple cases of multiple devices. *2017 2nd International Conference on Fog and Mobile Edge Computing, FMEC 2017*, (May), 235–240. https://doi.org/10.1109/FMEC.2017.7946437

Mun, D., Dinh, M. Le, & Kwon, Y. (2016). An Assessment of Internet of Things Protocols for Resource-Constrained Applications. https://doi.org/10.1109/COMPSAC.2016.51

Orgerie, A., & Lefevre, L. (2014). A Survey on Techniques for Improving the Energy Efficiency of Large Scale Distributed Systems, ACM Computing Surveys Volume *46 Issue* 4, Article 47, pp 1–35.

Paethong, P., Sato, M., & Namiki, M. (2016). Low-power Distributed NoSQL Database for loT Middleware. *2016 Fifth ICT International Student Project Conference (ICT-ISPC)*, Nakhon Pathom, 2016, pp. 158-161. doi: 10.1109/ICT-ISPC.2016.7519260

Priyanka, S. T. (2016). Performance Evaluation of Time series Databases based on Energy Consumption. Thesis no: MSEE-2016:35

Qiyao Xie, Tian Huang, Zhihai Zou, Liang Xia, Yongxin Zhu, & Jiang Jiang. (2012). An accurate power model for GPU processors. *Computing and Convergence Technology (ICCCT), 2012 7th International Conference On*, (July), 1141–1146.

Reda, S., & Nowroz, A. (2012). Power Modeling and Characterization of Computing Devices: A Survey. *Foundations and Trends® in Electronic Design Automation*, *6*(2), 121–216. https://doi.org/10.1561/1000000022

Satyanarayanan, M., Gibbons, P. B., Mummert, L., Pillai, P., Simoens, P., & Sukthankar, R. (2017). Cloudlet-based Just-in-Time Indexing of IoT Video. *Global Internet of Things Summit (GIoTS)*, Geneva, pp. 1-8. doi: 10.1109/GIOTS.2017.8016212

Satyanarayanan, M., Lewis, G., Morris, E., Simanta, S., Boleng, J., & Ha, K. (2013). The role of cloudlets in hostile environments. *IEEE Pervasive Computing*, *12*(4), 40–49. https://doi.org/10.1109/MPRV.2013.77

Shey, J., Rakvic, R., Ngo, H., Walker, O., Tedesso, T., Blanco, J. A., & Fairbanks, K. (2016). Inferring trimming activity of solid-state drives based on energy consumption. *Conference Record - IEEE Instrumentation and Measurement Technology Conference*, *2016–July*. https://doi.org/10.1109/I2MTC.2016.7520537

Song, S., Su, C., Rountree, B., & Cameron, K. W. (2013). A simplified and accurate model of power-performance efficiency on emergent GPU architectures. *Proceedings - IEEE 27th International Parallel and Distributed Processing Symposium, IPDPS 2013*, 673–686. https://doi.org/10.1109/IPDPS.2013.73

Stokke, K. R., Stensland, H. K., Halvorsen, P., & Griwodz, C. (2016). A High-Precision Power Model for the Tegra K1 CPU, GPU and RAM. Poster. *Proc of GTC*.

Sun, X., & Ansari, N. (2017). Latency Aware Workload Offloading in the Cloudlet Network. *IEEE Communications Letters*, *21*(7), 1481–1484. https://doi.org/10.1109/LCOMM.2017.2690678

Tso, F. P., White, D. R., Jouet, S., Singer, J., & Pezaros, D. P. (2013). The Glasgow Raspberry Pi Cloud : A Scale Model for Cloud Computing Infrastructures, (July). *2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops*, Philadelphia, PA, 2013, pp. 108-112. doi: 10.1109/ICDCSW.2013.25

Ukidave, Y., Ziabari, A. K., Mistry, P., Schirner, G., & Kaeli, D. (2014). Analyzing power efficiency of optimization techniques and algorithm design methods for applications on heterogeneous platforms. *International Journal of High Performance Computing Applications*, *28*(3), 319–334. https://doi.org/10.1177/1094342014526907

Weaver, V. M., Johnson, M., Kasichayanula, K., Ralph, J., Luszczek, P., Terpstra, D., & Moore, S. (n.d.). Measuring Energy and Power with PAPI. *2012 41st International Conference on Parallel Processing Workshops*, Pittsburgh, PA, 2012, pp. 262-268. doi: 10.1109/ICPPW.2012.39

Zhu, C., Leung, V. C. M., & Shu, L. E. I. (2015). Green Internet of Things for Smart World, 2151–2162. *IEEE Access*, vol. 3, pp. 2151-2162, 2015.
doi: 10.1109/ACCESS.2015.2497312