GEMS: Glasgow Energy Measurement System for the Raspberry Pi Cloud

Piotr Hosa

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

Level 4 Project — February 26, 2016

Abstract

Acknowledgements

Education Use Consent

Contents

Chapter 1

Introduction

1.1 Motivation

Cloud technology has become increasingly popular in recent years. Owners of large infrastructures such as Google, Amazon and Microsoft are able to offer their customers highly scalable services at a lowering cost and no upfront investment. However, dealing with considerable amounts of data and traffic invokes a need for sizeable hardware, which imposes various challenges on its owners. These challenges include automated service provisioning, virtual machine migration and energy management among others [6].

The Glasgow Raspberry Pi Cloud (RPC), as discussed in section 1.2, is a scale model data centre. The RPC is a research and educational tool and it aims to mimic full-sized cloud infrastructures. This is incrementally achieved by the projects done on the RPC, which use leading virtualisation and cloud software such as Docker and Kubernetes.

This project focuses on developing a power monitoring system for the RPC, which is a fundamental step toward power management. Monitoring makes it possible to identify machines with abnormal power consumption, but also to compare the power efficiency of similar software frameworks. While power management is not of particular concern in a cluster of this size, implementing such a system advances the project by making the RPC more similar to full-scale data centres. In the full-scale cases energy management is highly important. Not only is it crucial for reducing operational costs but also to keeping in accordance with environmental concerns, which will only increase in the future [1].

1.2 The Glasgow Raspberry Pi Cloud

1.2.1 Overview

The Raspberry Pi project at the University of Glasgow was started in 2012 by a group of researchers in the School of Computing Science. The motivation to pursue the project was a lack of satisfactory methods that would have allowed to conduct research on data centres and cloud computing. Before the Raspberry Pi Cloud, recreating such infrastructures relied mainly on software simulations and using physical environments with a very limited number of machines. Both of these methods were aimed at reducing costs of such research systems. The RPC eliminated the financial overhead by using 56 low-cost computers and therefore allowed to model full-scale data centres more closely. Using Raspberry Pis in the project also eliminated other challenges associated with traditional data centre servers like space constraints, cooling and considerable energy used by the machines. The RPC also has the advantage of exhibiting real network traffic and running real cloud services, which is difficult to model in software simulations [3].

1.2.2 The Original Cloud

In its original version the RPC consisted of 4 racks of 14 Raspberry Pi 1 Model B computers. Each rack had a top-of-rack switch that connected to a central OpenFlow switch and then to the Internet. There was also a master node also connected to the OpenFlow switch that for system management. Each of the nodes in the rack run the Raspbian version of Debian. The first software developed for the cluster was a REST API with a web interface. The project also involved experimenting with LXC virtualisation, Hadoop and Software Defined Networking [5].

1.2.3 More Recent Projects

One of the most recent projects with the RPC involved a hardware upgrade. One rack of 14 computers was fitted with Raspberry Pi 2 Model B devices, which are more powerful than the ones used previously. The new infrastructure extended the potential applications of the cluster and made it possible to experiment with Kubernetes, Google's cluster management software. In the recent months Kubernetes has been adapted to the ARM architecture of the Raspberry Pi [4] and work has been done on the Kubernetes Dashboard [2].

Before the upgrade, a team of students worked on a wake-on-LAN hardware and load balancer software for the cluster, which used Docker and SaltStack.

1.3 Related Work

Papers that deal with similar topics (links commented out):

1. PowerPi: Measuring and Modeling the Power Consumption of the Raspberry Pi
2. Affordable and Energy-Efficient Cloud Computing Clusters: The Bolzano Raspberry Pi Cloud Cluster Experiment
3. A Study Case of Restful Frameworks in Raspberry Pi: A Perfomance and Energy Overview
4. A Low-Cost Computer Cluster for High-Performance Computing Education
5. Iridis-pi: a low-cost, compact demonstration cluster
6. Impact of different compiler options on energy consumption
7. Energy Measurement Infrastructure

Chapter 2

Requirements

2.1 Use Cases

Users can

1. See overview of data from cluster in graphs/tables
2. See overview of cluster as heatmap
3. Obtain CSV files to analyse data in more detail

2.2 Functional Requirements

1. Table with requirements
2. User stories in appendix(?)

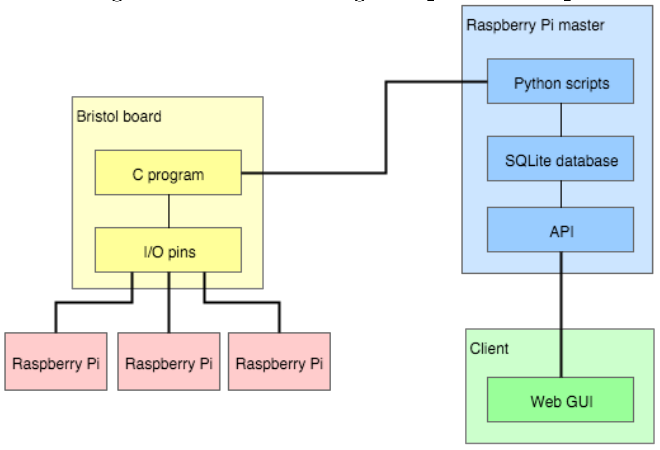
2.3 Non-Functional Requirements

1. Accessibility
2. Compatibility (plug and play)
3. Recoverability (broken connections, API down)
4. Performance (multiple users can access without over-stressing the system)

Chapter 3

Design

Figure 3.1: System diagram demonstrating the power component of the system.



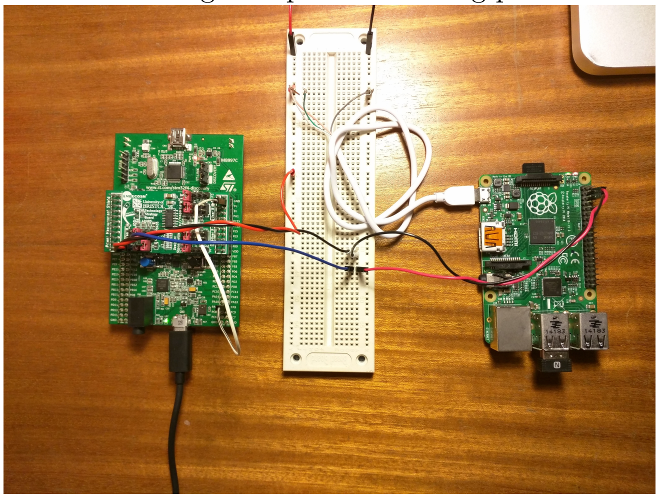
3.1 Hardware

1. Adapted from the MAGEEC project
2. The setup in the Glasgow PiCloud (MAGEEC wires + network)

3.2 API Server

1. Choice of OS for the Pi
2. Python chosen as it is most frequently used on the Pi and therefore there is a lot of support and documentation for it.
3. Flask (and Flask restless) chosen for API. Other options available (Django, Tastypy and Sadman) but Flask is simple and popular and therefore seemed like the right choice.
4. Chose to use SQLite dabase with Flask SQLalchemy to make database access easier.

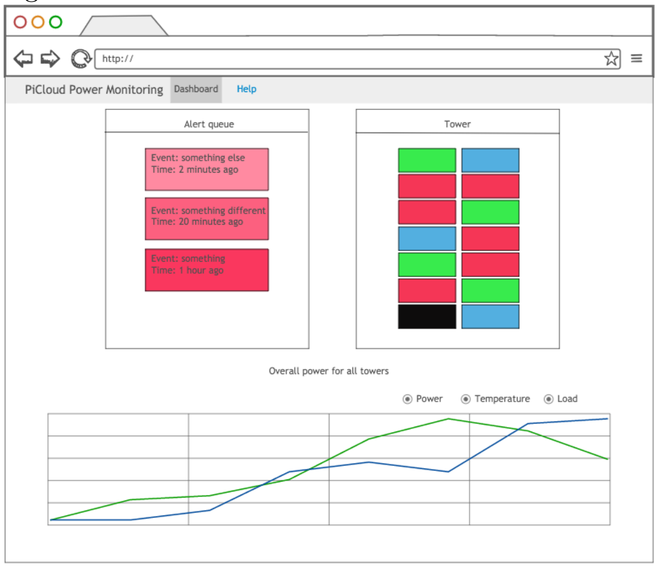
Figure 3.2: Basic wiring set-up for measuring power consumption.



3.3 Web Client

1. Single Page Application design (rendering and logic done in client which takes processing off server; more responsive websites)
2. There is a number of frameworks available (Ember.js, ExtJS, React, Knockout) but Angular has a large community and it supports bidirectional data binding, which some of the other frameworks lack
3. Many front-end frameworks have been adapted to Angular but if they have not it is possible to do it by hand.

Figure 3.3: Dashboard wire frame for the web client



Event: something else
Time: 2 minutes ago

Event: something different
Time: 20 minutes ago

Event: something
Time: 1 hour ago

Chapter 4

Implementation

4.1 Logger Node

1. Logging python (based on MAGEEC pyenergy library)
2. Database and API
3. Web client
4. Salt managing scripts

4.2 Minion Nodes

1. Reporting python

Chapter 5

Evaluation

5.1 Testing

1. Python tests for scripts
2. JavaScript tests(?)

5.2 Acceptance Testing

1. Overview of testing scenario
2. Feedback from testers
3. Changes in system based on feedback

Chapter 6

Conclusion

6.1 Summary

1. What has been implemented
2. What the system can do
3. Comment on usability

6.2 Future Work

1. What other features are the next logical step
2. How do those features fit within the technology stack

Appendices

Appendix A

Appendix

Bibliography

[1] Mehiar Dabbagh, Bechir Hamdaoui, Mohsen Guizani, and Ammar Rayes. Toward energy-efficient cloud computing: Prediction, consolidation, and overcommitment. *IEEE Network*, 29(2):56–61, 2015.

[2] H. McWha. Kubernetes Dashboard run as Docker image on Raspberry Pi 2., 2016.

[3] Fung Po Tso, David R. White, Simon Jouet, Jeremy Singer, and Dimitrios P. Pezaros. The Glasgow Raspberry Pi Cloud: A scale model for cloud computing infrastructures. *2013 IEEE 33rd International Conference on Distributed Computing Systems Workshops*, 2013.

[4] Jim Walker. How to: Kubernetes multi-node on Raspberry Pi 2s, 2015.

[5] David R. White. A Raspberry Pi cloud, 2013.

[6] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *J Internet Serv Appl*, 1(1):7–18, 2010.