Open Access

Open Journal of Cloud Computing (OJCC)
Volume 5, Issue 1, 2018
http://www.ronpub.com/ojcc
ISSN 2199-1987

A Lightweight Network-Controlled Power Strip for Low-Cost Cluster Systems

Henry-Norbert Cocos, Christian Baun

Faculty of Computer Science and Engineering, Frankfurt University of Applied Sciences,
Nibelungenplatz 1, 60318 Frankfurt am Main, Germany,
cocos@stud.fra-uas.de, christianbaun@fb2.fra-uas.de

Abstract

Low-cost clusters are not equipped with costly, sophisticated tools and cannot be controlled remotely. This work aims at addressing this issue and develops a lightweight network-controlled power strip, which enables administrators to monitor the cluster and perform operation via remote. The power strip is controlled via a web interface and a RESTful web service, which are implemented with the programming language Python and the web framework Flask. The solution is inexpensive and easy to implement and use. In this paper, we describe in detail the development and construction of the prototype of the solution and discuss its purchase cost and power consumption.

Type of Paper and Keywords

Short communication: *network controlled power strips, clusters, single board computers, Web services*

1 Introduction

The hardware that is used to provide cloud infrastructure and platform services are typically cluster systems, which implement parallel computing with distributed memory. Cluster systems usually consist of server systems as nodes, which are equipped with powerful administration and maintenance tools like the HPE Integrated Lights-Out (iLO) [13] or IBM Integrated Management Module (IMM) [14]. These tools allow to power a system on and off via remote, no matter in which state this system currently is.

However, if clusters are constructed by using commodity hardware systems or even lesser expensive components like single board computers, such cluster systems lack advanced tools of administration and maintenance such as iLO or IMM. The administrators can not detect defects and perform operations on the cluster remotely and they need to work on site with the cluster to investigate and fix problems and switch crashed nodes on or off.

Clusters of single board computers have less purchase cost and are easy to operate, and therefore they are a good option for education and research projects with less funding [2]. The Faculty of Computer Science and Engineering of the Frankfurt University of Applied Sciences possesses a cluster of 128 nodes of Raspberry Pi 3 single board computers with 512 physical CPU cores [6]. This cluster serves as a testbed for scientific research and education, and gives students the opportunity to experiment with different parallel computing models.

The Raspberry Pi 3 computer provides very limited hardware resources compared with servers or workstations that consist of commodity hardware components. It is equipped with a CPU with four cores (ARM Cortex A8), 1 GB of main memory and a 10/100 Mbit Ethernet interface. Those characteristics show the hardware limitations of the system but yet it is capable of running different Linux distributions.

Table 1: The characteristics of the power strips on the market and our prototype

	Brennenstuhl Premium-Web-Line [8]	**ALLNET ALL4176** [1]	**Our Prototype**
Interfaces	LAN	LAN, WLAN, USB	LAN, Wi-Fi, USB, Bluetooth
Interaction	main switch, webpage	switch for sockets, webpage	main switch, taster, webpage, webservice
Software	not adaptable	not adaptable	adaptable
No. Sockets	5 (2 permanent)	6	9
max. Current	16 A	10 A	10 A
Price	129 €	269 €	160 €

The hardware resources are also sufficient to use them for running object-based cloud storage services [4][5] and for the investigation of the scalability of parallel computation tasks [3].

With a cluster of low-cost nodes like Raspberry Pi 3 computers, the crash of nodes during operation is a common event. Since lack to powerful, costly maintenance tools, the node crash can not be detected and fixed remotely and the administrators have to work on site. In order to address this issue we develop in the paper an inexpensive tool, a lightweight network-controlled power strip, which monitors the nodes of cluster and enables administrators to power crashed nodes on and off remotely. The network-controlled power strip consists of a power strip (hardware part) and a web application and web service (software part). The web application and service monitor the nodes of the cluster and allow users to reset the crashed node over networks [9]. The run of nodes is monitored by the command-line tool `ping`. The reply of `ping` is used to detect whether a node is still in operation state or has failed. An externally attached optocoupler circuit is used to reset the failed Raspberry Pi node. The development and construction of the prototype of the network-controlled power strip will be presented in detail in this paper.

Network controlled power strips offer a comfortable and time-saving way to control connected appliances. The technology of network controlled power strips is also interesting in regard of the growing field of smarthome appliances and home automation scenarios. The prototype presented in this work is easy to construct and the adoption of a Raspberry Pi as the central controlling component make it very flexible in operation. Users can build such a power strip easily for their home environment with low costs and the potential of extensibility. Its application is not only limited to the control of cluster systems, and automation scenarios are an example of other possible applications

The rest part of the paper is organized as follows. We first discuss related work in Section 2. Section 3 presents our lightweight network-controlled Power Strip. It describes in detail the architecture of hardware and software and the development and construction of the prototype, and reports the primary use experiences of the prototype. The conclusions of the work are given and the future work is discussed in Section 4.

2 Related Work

A large number of network controlled power strips exist on the market. They are usually listed under the term "smart switch" or "smart power strip". These smart power strips provide different functionality and their prices vary from 60 € to 269 € [1]. Table 1 shows the characteristics of two popular power strips on the market and their purchase price. For example, ALL4176 is a smart power strip from the hardware vendor Allnet [1], and it is equipped with six individually switchable plugs and hardware buttons to switch the individual plugs on and a Wi-Fi and Ethernet interface. Its current market price is 269 €. Table 1 also compares the two products with our prototype of power strip.

These power strips are usually built by using a microcontroller for the access of the sockets, and the microcontroller also provides a web server for Internet access. This offers the users the functionality to control the individual sockets via remote. However, the fact that the web server is built into a microcontoller is a serious drawback, because the used web server software is not interchangeable and can not be adapted to further user needs. The web server is implemented inside a closed system and it cannot be configured by users. Therefore, it is not as flexible and adjustable as an open solution. Another drawback is that the web interface (webpages), which is offered by the power strips, is not usable in a scenario where a machine automatically communicates with the device. For this scenario, a RESTful web service would be beneficial and using a different web server software is desired.

Klein and Wenzel [15] developed a power strip. The power strip is also constructed by using a Raspberry Pi single board computer as the central component and attach it to relays, which are used to switch the build-in plugs. The work of Klein and Wenzel shows the opportunities of a self-made power strip solution and it demonstrates how to construct a smart power strip by using a Raspberry Pi. Different from our work, the focus of their work is on the home automation software OpenHAB [17]. Their setup enables the integration of the built solution into an existing home automation environment. The solution of Klein and Wenzel makes use of a wooden casing and is therefore (compared with our prototype) relatively large in its dimensions. Their solution also lacks different control methods. In their work, the build-in plugs are controlled only via command line but there is no work on the use of a web page or web service to interact with the device.

Lee et al. [16] built a smart power strip by using a microcontroller and relays. Their power strip is designed as an adapter between the wall outlet and the power strip line. This enables the adapter to act as a bridging device making connected devices controllable via a smartphone app. In contrast to our prototype, the adapter is only controllable via an Android App using the Internet of Things (IoT) platform software Blynk [7]. The design of the adapter is simplistic and lacks options to extend its functionality.

The investigation of the existing solutions led to the development and construction of our prototype, which focuses on small dimensions and connectivity. By using the Linux operating system Raspbian, it is possible to choose among different free web server software solutions. For controlling the power plugs in our power strip, the GPIO ports of the Raspberry Pi are used. The programming language Python is adopted to control the GPIO ports of the Raspberry Pi and the Python framework Flask is used to host a web interface and a RESTful web service.

3 Lightweight Network-controlled Power Strip

This section presents our network-controlled power strip, which aims at providing a lightweight tool for low-cost clusters. Firstly, an overview of the system is given in Subsection 3.1, where the relationship of the hardware and software components are described. The hardware components and the construction of the power strip are then elaborated in Subsection 3.2. In Subsection 3.3, the design and implementation of the software part are presented in detail. The last part of this section (Subsection 3.4) reposts our first observations and issues in the use of the prototype.

3.1 The System Overview

The network-controlled power strip allows users to control low-cost clusters remotely. Figure 1 gives an overview of system and illustrates the players and the communication relation between them. The web application provides a user interface and the web service is used for machine-to-machine. They provide users with the capability of performing remote control to the cluster. The design and development of the web application and the web service will be discussed in detail in Subsection 3.3.

The power strip is used to directly control the nodes of cluster. Its central component is a Raspberry Pi single-board computer. The web application communicates over networks with the RPi.GPIO library of the Raspbian operating system of the Raspberry Pi. This library is set on the boundary between the software and hardware parts and is the binding interface between the web service and the power strip. The power strip performs the operation on the cluster via the GPIO ports of the Raspberry Pi. The GPIO ports are connected to the sockets and communication between them is unidirectional. Via the sockets the power strip can control the cluster, but the cluster does not communicate with it. Subsection 3.2 provides a detailed look into the power strip.

3.2 Construction of the Power Strip

Figure 2 gives an overview of the hardware part. As showed in this figure, a junction box is used as an enclosure for the hardware components. The Raspberry Pi is the central controlling component of the power strip and connected to the network via a network cable. A button is connected to a GPIO port of the Raspberry Pi and can be used to start the sockets. The GPIO ports of the Raspberry Pi are connected to the relay card. Through those GPIO ports the relays of the relay card are controlled and the sockets are supplied with current.

Figure 2 shows major hardware components and provides a simplified architecture of hardware. The simplification aims to demonstrate the functionality of hardware and to enhance the understanding of the internal structure. A prototype of the power stripe is implemented based on the architect, and Figure 3 presents the prototype and offers a comprehensive view of the power strip. We now give a detailed look into the prototype.

As showed in Figure 3, our power strip consists of four major components:

- **A** - USB power supply for the Raspberry Pi and the relay card

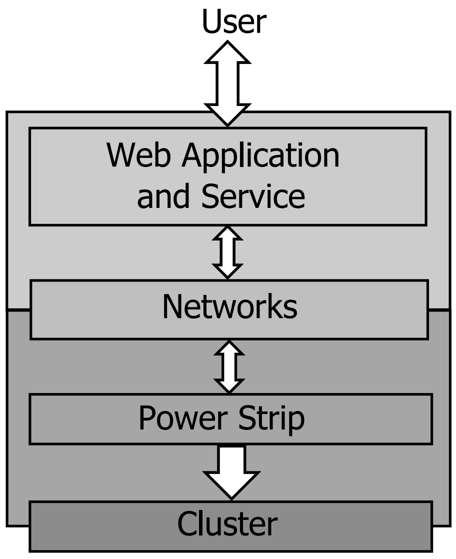

Figure 1: The system overview of the network-controlled power strip

Table 2: the hardware components used to construct the power strip and their purchase price

Component	**Quantity**	**Price**
Raspberry Pi 3	1	35 €
MicroSD card	1	7 €
USB cable	2	5 €
Junction box	1	25 €
Flush-mounted socket	10	30 €
Connecting cable 3x1,5 10m	2	10 €
Supply cable	1	7 €
Connection terminal	2	4 €
Small parts for assembly	-	10 €
Button	1	2 €
Switch	1	2 €
Status LED	1	1 €
Relay card	1	8 €
Female connector 40 pin	1	2 €
Anker USB power supply	1	12 €
Sum		160 €

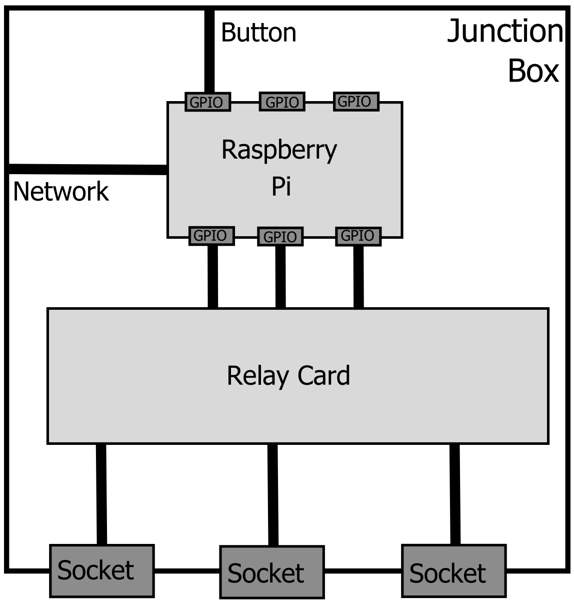

Figure 2: The architecture of the power strip

- **B** - Button for the sequential operation of sockets
- **C** - GPIO-Ports of the Raspberry Pi attached to the relay card
- **D** - Main switch of the system

For the housing of the hardware components, an electrical junction box is used as enclosure. This junction box is selected because it features pre-drilled holes which carry the power plugs. It is also suitable for carrying electrical components because it is made of a non conductive material and has a bolted cover to house the internal components. The power plugs are integrated into the predilled holes and are internally connected to a relay card via wires. The free space of the junction box is sufficient for up to 10 power plugs. One of the power plugs is used for the USB power supply (see the component A in Figure 3) of the internal Raspberry Pi computer and the relay card.

The relay card is used because the power plugs have an output voltage of 230 Volts. The GPIO ports of the Raspberry Pi are not able to switch such a voltage level. Therefore, a relay card is located between and connected to the Raspberry Pi and the power plugs. The used relays operate with a voltage of 230 Volts and a maximum current of 10 Ampere. The connection between the Raspberry Pi and the relay card is established with jumper cables. Figure 3 C shows the connection between the Raspberry Pi and the relay card.

Component B in Figure 3 is the button which starts the sequential power up of all sockets. The powerup is implemented in a sequential manner because powering up all sockets at the same time causes a load peak, which triggered the fuse of the electrical grid of our lab where the cluster is operated. Close to the button (the component B) is a LED, which is installed to inform about the state of the system. The color of the LED turns yellow when the Raspberry Pi is in ready state and did boot up. If the sockets are all in operation state, the LED

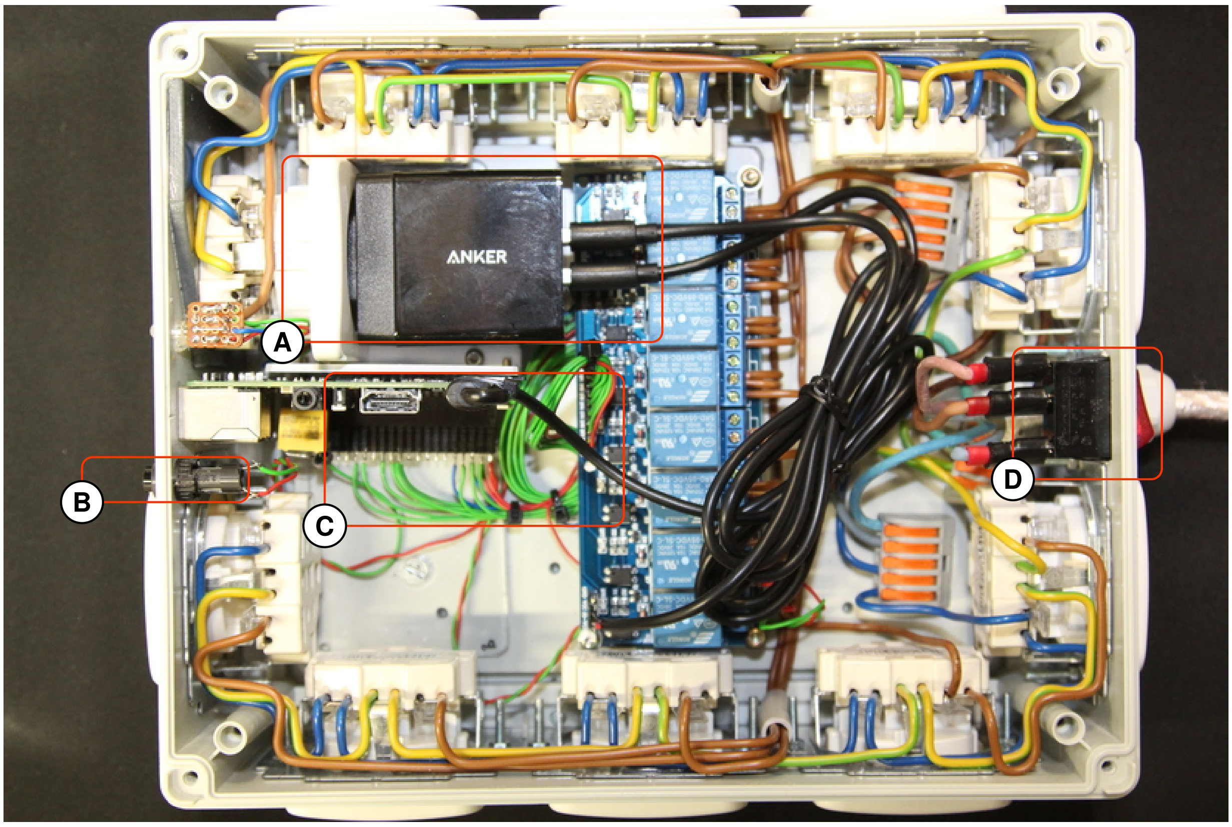

Figure 3: A prototype of the power strip

turns green. The LED is red in case the Raspberry Pi has not finished to boot up. The system also features a switch (see the component D in Figure 3) to startup the whole system.

Purchase Cost: Table 2 lists the hardware components used to construct the power strip and their purchase price. The total cost of all components used for the prototype is approximately 160 €. The list of components can be scaled up and down in order to meet the needs of the user. This prototype was built with the objective of controlling a large cluster. A detailed plan that shows the internal wiring of the prototype is presented in [10].

Power Consumption:

$$C_Y = \mathrm{E} \times\ 24\,\frac{\text{hour}}{\text{day}} \times\ 365.25\,\frac{\text{days}}{\text{year}} \times \frac{0.25\,€}{\text{kWh}} \qquad (1)$$

Equation (1) calculates the electric energy costs per year [2]. For the calculation of the electric energy costs of the Raspberry Pi 3, which is integrated into the prototype, we assume a power consumption of 2 Watts (E) and 0.25 € per kWh. As a result, the calculated electric energy costs per year (C_Y) for a round-the-clock (24/7) run of the prototype is 4,38 €. Combined with the purchase cost of 160 € for the prototype (see Table 2) the total cost of ownership is 164,38 € for each year. In comparison, the power consumption of a cluster with 128 Raspberry Pi 3 worker nodes is between 350 Watts in idle operation mode and 650 Watts during peak load [6]. The electric energy costs of such a cluster for a 24/7 operation is between approximately 767 € and 1,425 €. Therefore, the adding and run of our prototype does not add much costs to the cluster, but brings significant benefits for the users in regards of the administration and maintenance of the cluster.

Furthermore, the prototype was constructed in a way that no users can be harmed when they are using it. No conductive parts are located outside the housing and therefore the safety of the users is ensured.

3.3 Development of the Web Application and Service with Flask

In the prototype of power strip, a Raspberry Pi single board computer is used to control the relay card (and through it the power plugs) via its GPIO ports. Several different ways exist to control the GPIO ports of the

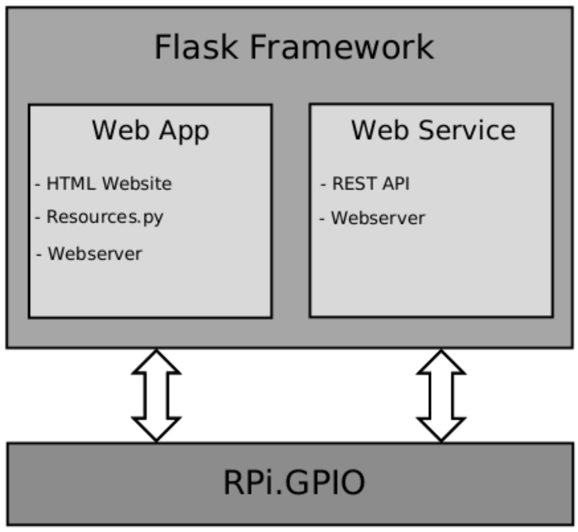

Figure 4: The overview of the Web application and Service

Raspberry Pi via software. One option is to interact with the GPIO ports by using shell scripts [11]. One benefit of this option is that it allows physical hardware access without the need for additional libraries. An alternative solution is using the WiringPi [22] framework, which allows to access and program the GPIO ports of the Raspberry Pi via the programming language C/C++. The library `RPi.GPIO` is another alternative solution for the programming language Python [19].

In order to provide a graphical user interface that allows the cluster administrators to interact with the prototype via remote, several web server software solutions have been investigated during this work. This resulted in using the web server framework Flask [18], which is a lightweight framework for web applications and is entirely developed in Python. With this framework, it is possible to host the web sever and build web services. These features make using Flask an ideal choice for the controlling of the power plugs. By using Flask to implement the web application and web service, the GPIO ports of the Raspberry Pi can be accessed from the same application by using the `RPi.GPIO` library.

Figure 4 presents an overview of the web application and the web service. The GPIO ports of the Raspberry Pi are physically connected with the relay card, which is in turn connected to the power plugs of the prototype. Table 3 presents the GPIO ports and their respective endpoints. The web application makes use of the GPIO numbering scheme as given in this table. The status-LED is connected with Relay 8. When the cluster is in the state of operation, the last relay (Relay 8) is switched to state on and the status-LED turns green. The GPIO port

Table 3: GPIO ports and the connected endpoints

GPIO port	Pin	Endpoint
GPIO 2	Pin 3	Relay 1
GPIO 3	Pin 5	Relay 2
GPIO 4	Pin 7	Relay 3
GPIO 5	Pin 29	Relay 4
GPIO 6	Pin 31	Relay 5
GPIO 7	Pin 26	Relay 6
GPIO 8	Pin 24	Relay 7
GPIO 9	Pin 21	Relay 8
GPIO 25	Pin 22	Status-LED
GPIO 26	Pin 37	Button

26 is configured as an input port and it detects the state of the connected button (see Table 3).

The software code and documentation of the web application and the web service are free available at `https://github.com/hcocos/Prototype_Socket/`.

3.3.1 Implementing the Web Application

Figure 4 presents the major components of the web application. The web application contains a website (client) and a sever. The web site provides a user interface implemented as webpages, which allows the users to control the power strip remotely. The web server communicates with the client and the GPIO ports of the Raspberry Pi in the power strip.

We use the web framework Flask for the development of the web application. Flask can be installed with all dependencies via the Python package management system `pip` with the command `sudo pip install flask`. After the installation of the packages, the server framework can be used in Python scripts. The web application is developed as a Python module that incorporates all the necessary resources for the application. The python script `resources.py` implements the access to the GPIO ports by using the `RPi.GPIO` library and runs the web server. The Python backend of the web application can access the GPIO ports and host the web pages at the same time, and this highlights the benefits of using the programming language Python. This design decision of using the Python language simplified the development and reduced the complexity of the web application.

The software of the web application is structured as follows:

- `\application_directory` - contains the entire web application

- `\application_directory\static` - contains the static data e.g. style sheets and images
- `\application_directory\templates` - contains the HTML pages of the application

The dynamic part of the HTML pages can be implemented as a Python script and therefore simplifies the access of the GPIO ports. The Python script accesses the GPIO ports by using the `RPi.GPIO` library. The static and dynamic HTML pages are rendered by using the function `render_template()` of the Flask framework.

Each webpage is accessed via a dedicated Python function. Additionally, a decorator [21] is implemented, which marks the endpoint of the web application and points to the location of the HTML page. The declarator `@webapplication.route('/')` points to the index HTML page of the web application (see the file `resource.py` in [10]). For different pages, different functions and decorators need to be implemented. The decorators can also be used for the purpose of user logging.

The web server is automatically started with the power strip by using the `supervisorctl` daemon [20].

3.3.2 Implementing the Web Service

In addition to implementing the web application, a RESTful web service is developed. The Web service is used for machine-to-machine communication via unique URLs (Uniform Resource Locator). One advantage of the RESTful web service is its their mechanism of communication because it adopts the most-commonly-used HTTP methods like `GET`, `POST`, `UPDATE` and `DELETE` for communication.

The Flask framework offers useful functions and data types for implementing RESTful web services. One example is the function `jsonify()`, which returns a JSON (Java Script Object Notation) object. These objects implement a structured data type and are serializable, which simplifies the communication between a sever and its clients.

Figure 5 presents the communication diagram of the web service. The first method of web service used is the status function, which returns the status of the power plugs. The method is accessed via the URL `/webservice/stem/status` by using the `GET` method. When invoked, the method returns a JSON object, which contains the status of the power plugs. Just like the web application, each Flask function is equipped with a decorator, which defines the endpoint of the web service and the access method used. The web service can be tested with the command-line tool `curl` [12].

```
HTTP/1.0 200 OK
Content - Type: application/json
Content - Length: 570
Server: Werkzeug/0.9.6 Python/3.4.2
Date: Wed, 28 Feb 2018 22:10:12 GMT
{   "Dose":
    { "2":
          { "name": "Steckdose Nr. 1",
            "status": 0 },
      "3":
          { "name": "Steckdose Nr. 2",
             "status": 0 },
      ...}
}
```

Listing 1: Excerpt of the reply of the web service

The following command issues a `GET` request on the web service

```
$ curl -i http://192.168.178.220:2789/
    webservice/stem/status
```

Listing 1 presents an excerpt of the reply of the web service behind the URL `/webservice/stem/status`. The status code inside the reply has value 200, which means that the request was successful. The reply also informs among others about the length of the payload and the software, which implements the web service. The payload contains the JSON object, which is requested from the prototype. This JSON object contains a list of power plugs and their current status.

The other methods of the web service, which have been implemented in this work, are used to switch single power plugs on and off. These methods respond with text messages instead of JSON objects because the switching of single power plugs is a triggered action and the response is only needed as a confirmation of this action.

The web service provide the capability of controlling multiple power strips from any clients, and help to orchestrate different power strips through a web interface. The functionality of the network-controlled power stripe can be expanded with additional methods of the web service in a simple way.

3.4 Primary Use Experiences

The first test was carried out with a small cluster that consists of six Raspberry Pi single board computers. The six Raspberry Pis were attached to a six-port USB power supply from the hardware vendor Anker with a rating of 60 Watts. The USB power supply of the cluster was attached to a socket of the prototype, and the power supply was tested on every socket. We tested the prototype using different cluster loads by increasing the number of running nodes in the cluster.

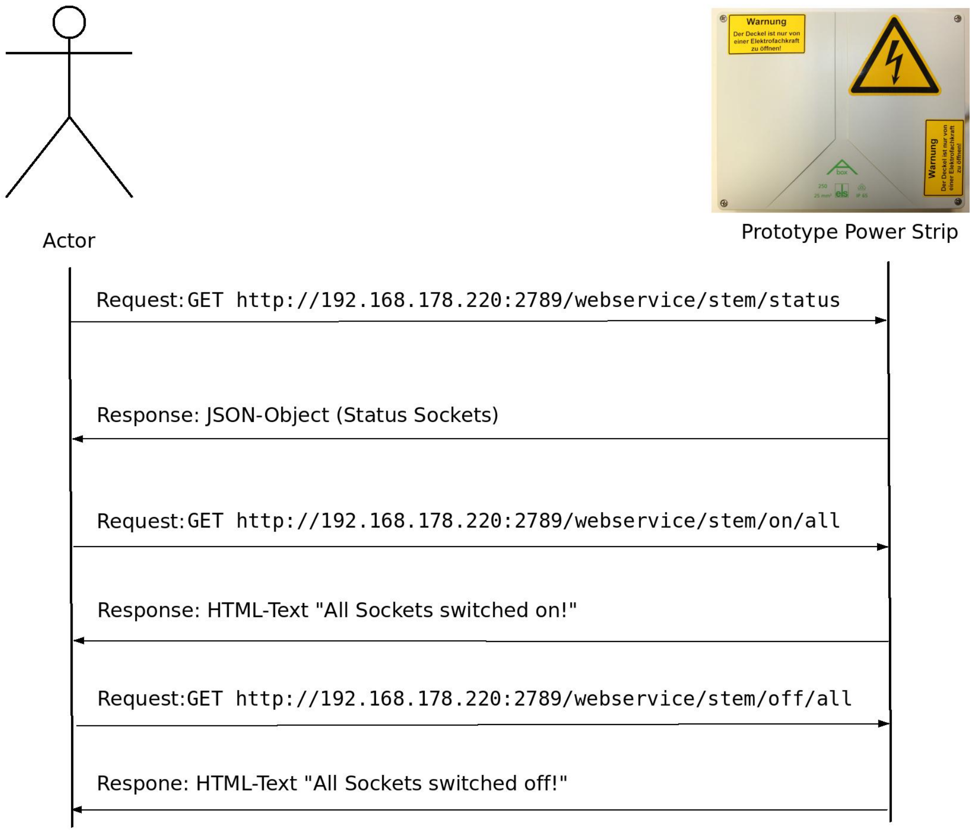
Actor
Prototype Power Strip
Warnung
Der Deckel ist nur von einer Elektrofachkraft zu öffnen!
Request:GET http://192.168.178.220:2789/webservice/stem/status
Response: JSON-Object (Status Sockets)
Request:GET http://192.168.178.220:2789/webservice/stem/on/all
Response: HTML-Text "All Sockets switched on!"
Request:GET http://192.168.178.220:2789/webservice/stem/off/all
Respone: HTML-Text "All Sockets switched off!"

Figure 5: The communication diagram of the web service

After the prototype was successfully tested on the small cluster, a larger cluster with 128 Raspberry Pis [6] is then used to further test the network-controlled power strip. The individual Raspberry Pis were attached to USB power supply with five and six USB ports. The USB power supplies were attached to ordinary power strip lines, which were in turn attached to the sockets of the prototype. With this setup, the power strip was tested under different load schemes of the cluster by changing the number of running cluster nodes.

The test lasted one month. In the period of time a broken relay occurred. An investigation of this error showed that the used relay card has a construction issue as it uses an optocoupler with a LED indicator on the input side. The built-in LED was designed for a different input voltage and the switching operation of the sockets resulted in an overload of the LED and therefore led to the destruction of the LED. The relays were not affected by this error and a replacement of the built-in LEDs with LEDs that support a higher voltage should cope with this issue.

4 Conclusions and Future Work

In this paper, we develop a lightweight network-controlled power strip and provide a solution to remotely control low-cost clusters that lack of costly, advanced administration tools. A prototype of the solution is implemented and tested in this work, and the prototype is inexpensive and easy to construct and operate. This hardware part (the power stripe) of this prototype was built with a Raspberry Pi single board computer as the central controlling component and its GPIO ports are used to control the other physical components. The Raspberry Pi switches the relays of a relay card, which operates the directly connected power plugs.

A web application and a RESTful web service are developed, which enable users to operate the power strip remotely. This software part was implemented in the programming language Python by using the Flask web framework. This framework offers useful functions for the development of web applications and web services and additionally offers a lightweight web server. The implemented Python application is also capable of accessing the GPIO ports of the Raspberry Pi via the `RPi.GPIO` library. The implemented RESTful web service allows machine-to-machine communication and is also implemented with the Flask framework.

The prototype was tested with different scale of clusters of single board computers, which are not equipped with advanced administration tools like HPE iLO or IBM IMM. The application of the prototype reduces the administration efforts of low-cost cluster systems. Our web application ansservice are implemented as free software and licensed under the terms of the GPLv3. Their source code and documentation can be found in the Git repository: `https://github.com/hcocos/Prototype_Socket/`

Future Work: One of the next steps is to investigate the durability of the prototype. Therefore, suitable test cases need to be designed and different charging states need to be evaluated. Only longterm operation can show how reliable the constructed prototype is. Furthermore the functionality of the web application will be improved further. In the current implementation, every user has full access to the webpage and the web service and therefore can switch the cluster off during operation. This need to be disabled by a proper logging function and the implementation of user accounts. Another relevant topic, which needs to be addressed, is the synchronization of the operation modes. In its current state, the power plugs can be activated by pressing the button of the junction box, while in the mean time, a remote user could switch them off in parallel.

The prototype was connected with the local network infrastructure by using the Ethernet interface of the Raspberry Pi. Recent revisions of the Raspberry Pi also offer a Wi-Fi interface. One of the next steps is to evaluate how reliable and fast a Wi-Fi connection would be without using an external antenna. Another interesting functionality, which may be implemented into the prototype, is the ability to measure the power consumption of the devices, which are connected to the power plugs and log the electric energy consumed by the devices.

Acknowledgements

This work is partly funded by the Faculty of Computer Science and Engineering Science of the Frankfurt University of Applied Sciences in the framework of 'Innovationsfonds Forschung' (IFOFO).

Many thanks to Rosa-Maria Spanou for her assistance in improving the quality of the paper.

References

[1] "Allnet all4176 power strip," http://www.allnet.de/en/allnet-brand/produkte/neuheiten/p/allnet-all4176-ip-steckdosenleiste-6-fach-schaltbar-per-netzwerk/, accessed 27th March 2018.

[2] C. Baun, "Mobile clusters of single board computers: an option for providing resources to student projects and researchers," *SpringerPlus*, vol. 5, no. 1, 2016.

[3] C. Baun, "Parallel image computation in clusters with task-distributor," *SpringerPlus*, vol. 5, no. 1, p. 632, May 2016.

[4] C. Baun, H.-N. Cocos, and R.-M. Spanou, "Ossperf – a lightweight solution for the performance evaluation of object-based cloud storage services," *Journal of Cloud Computing*, vol. 6, no. 1, p. 24, Dec 2017.

[5] C. Baun, H.-N. Cocos, and R.-M. Spanou, "Performance aspects of object-based storage services on single board computers," *Open Journal of Cloud Computing (OJCC)*, vol. 4, no. 1, pp. 1–16, 2017. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:101:1-2017100112204

[6] C. Baun, H.-N. Cocos, and R.-M. Spanou, "Erfahrungen beim Aufbau von großen Clustern aus Einplatinencomputern für Forschung und Lehre," *Informatik-Spektrum*, vol. 41, no. 3, 2018, https://link.springer.com/article/10.1007/s00287-017-1083-9.

[7] Blynk, "Democratizing the internet of things," https://www.blynk.io, accessed 20th May 2018.

[8] Brennenstuhl, "Premium-Web-Line Internet Extension Socket 5-way," https://www.brennenstuhl.com/en-DE/products/extension-leads/premium-web-line-internet-extension-socket-5-way-black-3m-h05vv-f-3g1-5-1-master-2-slave-2-permanent, accessed 29th April 2018.

[9] H.-N. Cocos, "Git repository automated reset," https://github.com/hcocos/Reset_Pi, accessed 27th March 2018.

[10] H.-N. Cocos, "Git repository prototype power strip," https://github.com/hcocos/Prototype_Socket, accessed 28th March 2018.

[11] elinux, "Rpi gpio code samples," https://elinux.org/RPi_GPIO_Code_Samples, accessed 30th March 2018.

[12] GibBook, "Everything curl," https://ec.haxx.se/, accessed 20th May 2018.

[13] HPE, "HPE iLO 4 User Guide," Hewlett Packard Enterprise, , 2018.

[14] IBM, "Integrated Management Module – Users Guide," International Business Manchines Corporation, , 2010.

[15] T. Klein and D. Wenzel, "Schaltbare steckdosenleiste mit dem raspberry pi," https://klenzel.de/5190, accessed 27th March 2018.

[16] Y. Lee, J. Jiang, G. Underwood, A. Sanders, and M. Osborne, "Smart power-strip: Home automation by bringing outlets into the IoT," in *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*, Oct 2017, pp. 127–130.

[17] OpenHAB, "Empowering the smart home," https://www.openhab.org, accessed 20th May 2018.

[18] A. Ronacher, "Flask web development, one drop at a time," http://flask.pocoo.org, accessed 30th March 2018.

[19] SourceForge, "Rpi.gpio python module," https://sourceforge.net/p/raspberry-gpio-python/wiki/Home/, accessed 30th March 2018.

[20] supervisord.org, "Supervisor: A process control system," http://supervisord.org/index.html, accessed 30th March 2018.

[21] P. Team, "View Decorators," http://flask.pocoo.org/docs/1.0/patterns/viewdecorators/, accessed 20th May 2018.

[22] WiringPi, "Gpio interface library for the raspberry pi," http://wiringpi.com, accessed 20th May 2018.

Author Biographies

Henry-Norbert Cocos studies computer science at the Frankfurt University of Applied Sciences. He earned his Bachelor degree in Computer Science in 2018 from the Frankfurt University of Applied Sciences. His research interest includes distributed systems and single board computers. Currently, he analyzes a 128 node cluster of Raspberry Pi 3 nodes in regard of different parallel computation tasks. For this work, he analyzes which administration tasks need to be carried out during the deployment and operation phase and how these tasks can be automated.

Dr. Christian Baun is a Professor at the Faculty of Computer Science and Engineering of the Frankfurt University of Applied Sciences in Frankfurt am Main, Germany. He earned his Diploma degree in Computer Science in 2005 and his Master degree in 2006 from the Mannheim University of Applied Sciences. In 2011, he earned his Doctor degree from the University of Hamburg. He is author of several books, articles and research papers. His research interest includes operating systems, distributed systems and computer networks.