PiRack

Philippe DIEP, Akram EL FADIL, Alexandre MESLET, Aude PLANCHAMP, Héloïse ROSTAN, Arian SENIOR

{phdiep, aelfadil, ameslet, aplanchamp, hrostan, asenior} @enseirb-matmeca.fr

Abstract. This paper deals with the solution to face the problem of managing a cluster of Raspberry Pis. A solution named PiRack, using 49 Raspberry Pis and a Master/Slave architecture has been developed. The PiRack set up, its architecture and the management tools developed are presented successively bellows.

1 Introduction

The Raspberry Pi Foundation [1] is a charity founded by David Braben, Eben Upton, Rob Mullins, Jack Lang, Alan Mycroft and Pete Lomas. Its aims at promoting the study of computer science around the world. The foundation has developed, among other innovations, a single-board computer called Raspberry Pi. This device has two main features, it is credit-card sized and low-cost.

This very low price can be explained by the fact that it is sold as just a card, without keyboard, mouse, power supply, screen, or housing. There are currently two existing models since the first launch in 2012, the oldest version costs nearby 30$ whereas the latest one is around 40$. Although the main ambition of the foundation was to foster computer science learning, this technology met a great success in the tech community. As a matter of fact, countless projects using those entities came to light especially in the following fields: video games, robotics, embedded systems and especially domotics.

Figure 1 - A Raspberry Pi 2

Furthermore, many scientists are interested in Raspberry Pi in order to create a cluster of those machines. In fact, gathering many independent computers in cluster is very interesting to get higher performance of computation. It allows parallelize computation and increase the number of resources available.

Moreover, the use of Raspberry Pi cluster is very useful for testing distributed algorithms. The distributed algorithms are a part of computer science that consists on a set of computers which are independent but seems to be a unique consistent computer for the users. Each entity, called nodes, is autonomous, makes local computation and works on the achievement of a common goal with the other entities.

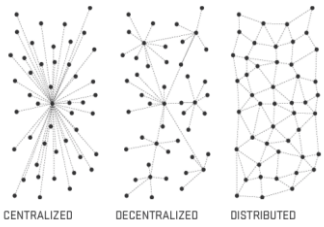

Figure 2 - Different types of computation systems

The European project Dionasys [2] falls within this context, with the aim of developing a network of Raspberry Pi clusters scattered all over Europe for testing Raspberry Pi at a very large scale. However, the clusters are likely to raise many issues as the multiplication of nodes increases the difficulty to manage them.

For instance, each node needs to be electrically powered, and all the nodes should be able to communicate with at least one another. This complexity increases the number of wires inside a cluster and results in a hardly manageable and maintainable clusters.

Besides, a generic solution hasn't still been found to store in a proper way the Raspberry Pis inside a cluster as the size of those entities depends on the one who build it and the computation level needed.

Therefore, the problem is to find a way to interconnect Raspberry pi in a cluster which can be easily maintained.

2 Project Context

The project aims at developing a tool to manage a cluster composed of 49 Raspberry Pi named the PiRack. This cluster is organized in twelve stacks of four Pis. As there is currently no existing interface which allows a user to simply manipulate a PiRack, one was created to enable any user to interact with this machine and administrate it.

Therefore, stacks located in the cluster can be visualized in two-dimensions, and pieces of information such as the IP or MAC address of a Raspberry Pi can be accessed. Besides, ping requests are used to know if the Raspberry Pi is either up or down. Thus, the user is informed of any breakdown and can fix the corresponding Pi. In the model developed, ping requests are actually sent periodically to determine each Pi's state. Eventually, as the cluster is organized in twelve stacks of four Raspberry Pis, a functionality was added to enable the user to get information from one Raspberry Pi as well as from stacks.

This system can be installed on any cluster composed of forty-nine Raspberry Pis and offers the primary features to operate it. Besides, as it is an open source project, the user is able to add new functionalities to send new requests to the Raspberry Pis.

3 PiRack buildings

3.1 PiRack architecture

Because the PiRack is an entity designed by multiple Raspberry Pis, a particular architecture has been set to control each computer unit: the master-slave model [3]. In computer networking, this model is represented by one process or device (known as the Master) that controls one or more processes or devices (known as slaves). In this devices relationship, there is a unique control flow direction which always starts from the Master to the slave(s).

In this project, a Raspberry Pi has been chosen as the Master to control the 48 other boards. The Master is connected to a relay module which provides power supply to each slaves. Finally, the slaves are organised by 12 stacks of 4 boards each.

The following describes how operates each component:

- A Raspberry Pi Master sends control commands to the slaves and monitor the PiRack through a web interface;
- Forty-eight Raspberry Pis slaves execute commands sent by the Master (retrieve CPU load, temperature, free memory space, etc.). These are the computation units of the PiRack;
- Switches and Ethernet wires connect each Raspberry Pi to the same local network;
- Relay module is composed of controllable electronic switches. Each switch provides power to only one stack of the Raspberry Pi.

3.2 PiRack configuration

Operating System installation. To set up the software environment ruling the PiRack operations, a specific Raspberry Pi operating system is required. The one used in this project (Raspbian Jessie 4.1[4] or higher) has been downloaded from the official website[1]. For the details of the operations made, follow the instructions attached this report.

In order to enable the Master Raspberry Pi to control the slave configuration, SSH (Secure Shell) keys have been generated (one private and one public key). Adding the public key to the */root/.ssh/authorized_keys* file (on the slave side) let further SSH connection to be made from the Master to the slaves, without any manual authentication. Eventually, in order to avoid the repetition of these operations on each slave SD card, a cloned image of the operating system configuration has been duplicated to every single Raspberry Pi manually.

The Master Raspberry Pi, which plays a particular role in the master-slave configuration supports some additional functionalities compared to the slaves. It must host a Python3 web server as well as a DHCP (Dynamic Host Configuration Protocol) server which distributes IP addresses to the slaves [5].

PiRack configuration settings. This section describes the operations that set the parameters required during a master-slave communication. In order to give orders to the slaves, the Master Raspberry Pi needs to know each machine IP address. Thus, a DHCP (Dynamic Host Configuration Protocol) server (*isc-dhcp-server*) has been established into the Master which role is to distribute IP addresses to nodes connected to the local network by listening on the *eth0* network interface. The figure below describes the simple local network used for the PiRack.

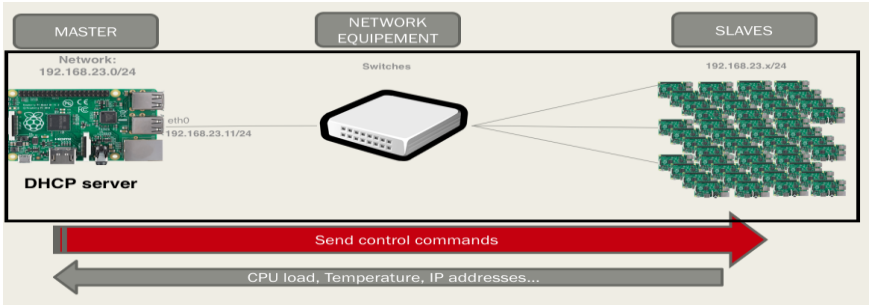

Figure 3 - PiRack local network

During the installation phase, IP addresses need to be attributed by the DHCP server to the slaves. A Python script handles the following functions:

1. Powering on/off the PiRack different stacks one by one;
2. Real-time monitoring the */var/log/syslog* file;
3. Applying new settings on the */etc/dhcp/dhcpd.conf* file.

A relay module has been added to the previous configuration to provide Raspberry Pi stacks with power supply.

PiRack installation function descriptions.

1. To manage the power supply distribution among stacks, relay modules have been added into the PiRack, interfacing the Master GPIO (General Purpose Input Output) pins with the slave's ones. In order to connect the Master Raspberry Pi with the relay module, a Master GPIO pin must be set as an OUTPUT, wiring a GPIO pin relay mod-

[1] https://www.raspberrypi.org/downloads/raspbian/

ule switch. Moreover, Master GPIO pins of 5V and GND (ground) have to be linked to the relay module GPIO pin of power supply.
When the Python script set the Master OUTPUT GPIO to the HIGH status, the corresponding relay module switch is triggered so that the module provides a stack with a 5V electrical supply. The latter is powering up. To power off a stack, the LOW status must be written into the Master GPIO OUTPUT pin. During the installation phase, stacks are power supplied one by one.

2. Monitoring the DHCP logs allows to detect when a DHCPREQUEST occurs (DHCP client requesting for an IP address to the server). The body of a DHCPREQUEST contains the client MAC address needed to be retrieved after powering on a stack. When a new MAC address is detected, it is added to a Python dictionary variable gathering *Rasp* objects. Then, Python script sets the following information for each Raspberry Pi: the MAC address, the IP address (which is incremented during Raspberry Pis detection) and the stack it belongs to.

3. Once the two previous functions have been triggered, a new */etc/dhcp/dhcpd.conf* file is written using each IP and MAC Raspberry Pi addresses stored in the dictionary variable. The given file then substitutes to the old DHCP configuration file and distributes now static IP addresses according DHCP client MAC addresses in the local network.

3.3 The Raspberry Pi Master

Web Client. As it was mentioned before, the idea of this project was to create an administration system to control a PiRack. This administration is made through a website and the user can simply interact with it without needing to understand the mechanisms triggered by its actions.
The structure of the website is displayed in the following scheme, the most important elements are the “Manage” page and the “Install” page. The “Install” page is where the installation of each Raspberry Pi in the PiRack is triggered through a unique button.

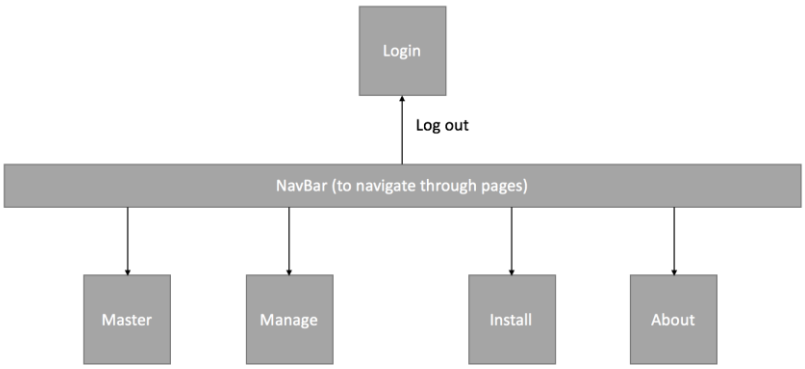

Figure 4 - Interface architecture

The manage page allows users to handle the PiRack:

- Visualise every Raspberry Pi ordered by stacks with their status;
- Order actions either on a Raspberry Pi or on a stack (ping, ask for temperature, reboot, shutdown);
- Switch view mode to a 2D view displaying the stacks of PiRack.

The current pages of the web Interface can be seen in the annexe.

AngularJS. The website was developed in HTML, JavaScript and CSS and it was decided to use the AngularJS framework [6] to develop its functionalities. It is currently the most popular and well documented framework.

Regarding the architecture of the website, AngularJS is a MVC (Model View Controller) [7] based framework. The idea behind MVC is that there is a clear separation in the code between managing its data (Model), the application logic (Controller), and presenting the data to the user (View).
The View gets data from the Model and displays it to the user. When a user interacts with the application by clicking or typing, the Controller responds by changing data in the Model. Finally, the Model notifies the View that a change has occurred so that it can update what it displays.

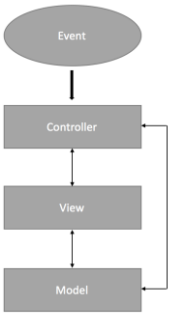

Figure 5 – MVC architecture

In AngularJS applications, the View is the Document Object Model (DOM) (HTML DOM [8]), the Controllers are JavaScript classes, and the Model data is stored in object properties.
The communication is made through directives, those are markers on a DOM element (such as an attribute, element name, comment or CSS class) that tell AngularJS HTML compiler to attach a specified behaviour to that DOM element (e.g. via event listeners), or even to transform the DOM element and its children.
Applied to the project, the MVC pattern is visible by looking at the directory structure you can see below.

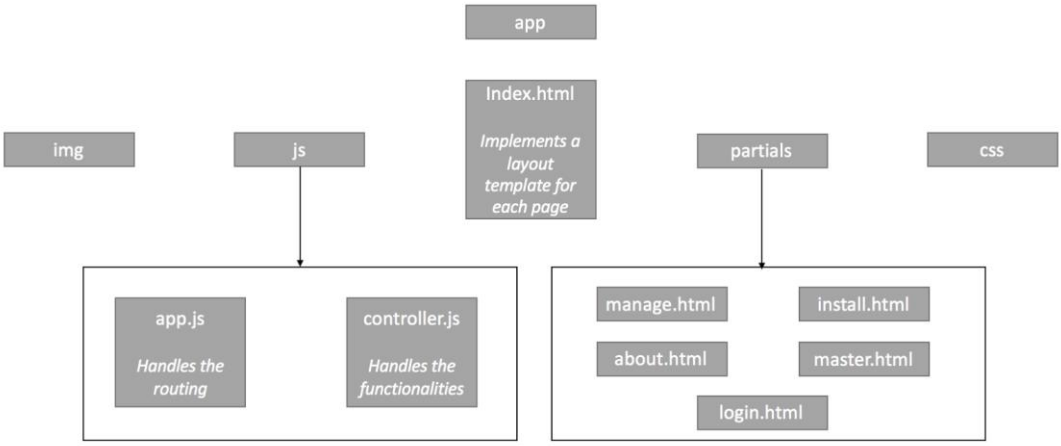

Figure 6 – Structure of the application

User Interface. AngularJS was used to develop functionalities, but then another framework was needed to develop the User Interface itself. Bootstrap [9] is the front-end framework commonly used with AngularJS. It is designed to kick-start the front-end development of web apps and sites. Among other features, it includes base CSS and HTML for typography, icons, forms, buttons, tables, layout grids, and navigation, along with custom-built jQuery-plugins and support for responsive layouts.

You can consider it as a library of functions or classes that enables to implement a responsive, good looking and most of all user-friendly front-end interface.

Regarding the architecture of the project, this is the edge of the first block in the Master, this web client then communicates with the second block (server) using a REST API [10] which will be detailed in the following part.

Web Server. The client communicates with the server through a REST API using HTTP requests. The idea was to focus on designing an API using a list of design rules that make it understandable and easy-to-use for future users.

The main features of this API are:

- Stateless design: each request includes all information needed by the server and the client;
- Self-descriptive messages: requests and responses are easily understood after spending minimal time reading the documentation;
- Semantic: The API uses existing features of the HTTP protocol to improve the semantic of input and output such as HTTP Verbs, HTTP Status.

The API's construction has been done step by step in a way that fits the customer needs. These steps are organized as following.

- Output formats: The most important thing to look at, when determining what format the API should output data in, is what users of the API would be using the data for and with. JSON (JavaScript object notation) was chosen for the API's output, giving the fact that it is a lightweight self-describing data-interchange format**.**

- URL structure: The URL structure is one of the most important steps of the API designing. The right endpoint names need to be defined in order to make the API much easier to understand and more predictable. Short and descriptive URLs were used and utilized to the natural hierarchy of the path structure.

Here are some possible endpoints:
- */api/v1.0/rasps*: Return all the Raspberry Pis of the PiRack;
- */api/v1.0/rasps/1*: Return the store that has id 1;
- */api/v1.0/rasps/options*: Return all the actions allowed on a Raspberry Pi;
- */api/v1.0/rasps/execute*: apply an action on a Raspberry Pi.

There are different kinds of errors, which were needed to be handled in the API, including permission errors, validation errors, not found errors, or even internal server errors. Therefore, a semantic HTTP status code is always returned with the response and an error message is sent if necessary with a more detailed description of what happened.

In addition, to return stored information about the Raspberry Pis, the web server needs to get some other piece of information directly from the slaves. Hence, requests and responses to the slaves had to be handled asynchronously in the Master part to avoid any blocking state that could crash the application. In order to do so, an API was built using a Python framework named Tornado [11]. Tornado is a Python web framework which provides an asynchronous networking library. By using non-blocking network I/O, Tornado can scale from tens to thousands of opened connections, making it ideal for long polling like Web Sockets, and other applications that require a long-lived connection to each user. The communication between the web server and the slaves is handled by a control server developed in Python which goal is to send actions to the slaves, get the response, process the data and then return the right JSON object.

3.4 Slaves

Communication over a network works as a postal communication. To reach a person, an address and a number must be specified. To ensure communication between the distinct components of the network architecture, static IP addresses are first delivered to every slave by the DHCP server implemented on the Master. After the slaves addressing process is achieved, a specific port number is selected by both the Master and the slave. In this way, every slave is connected to a port using an information transport protocol. To support this transfer of information, two network protocols were exhibited: UDP (User Datagram Protocol) [12] and TCP (Transport Control Protocol) [13]. At the beginning of the project, the network security was planned to be established at a very earlier stage by the I2C network thanks to Ethernet links between Raspberry Pis. Therefore, UDP was chosen over TCP as it was less security-demanding and easier to develop. As a matter of fact, contrary to the TCP protocol, UDP doesn't establish sessions between the Master and the slaves to enable them to communicate. Thanks to the connections between the slaves and the Master, this one is able to send queries to them in order to determine several inherent parameters such as the Raspberry Pi's temperature or the CPU load. These queries lead to the execution of bash commands which aliases are gathered in a unique configuration file (*.ini*). The format of this file is described on the lines below:

```
[Section] Command Alias = Bash command
```

For instance:

```
[Stack] CPU = cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
```

Every time a slave receives a query from the Master, the configuration.ini file is firstly parsed in order to be readable by slaves. Then, the configuration file is browsed until the command alias corresponding to the one received is found. When there is a match, the query-aimed slave executes the identified command using the system C function. The result of this process is then written into a temporary text file which is subsequently read to be sent to the Master. Once this process completed, the temporary file is deleted and the slave gets back to its initial state. In this manner, the state of each slave can be determined, and thus, the maintenance of the PiRack can continuously be assured. In the same way as Raspberry Pis, a project team is quite more efficient when managed.

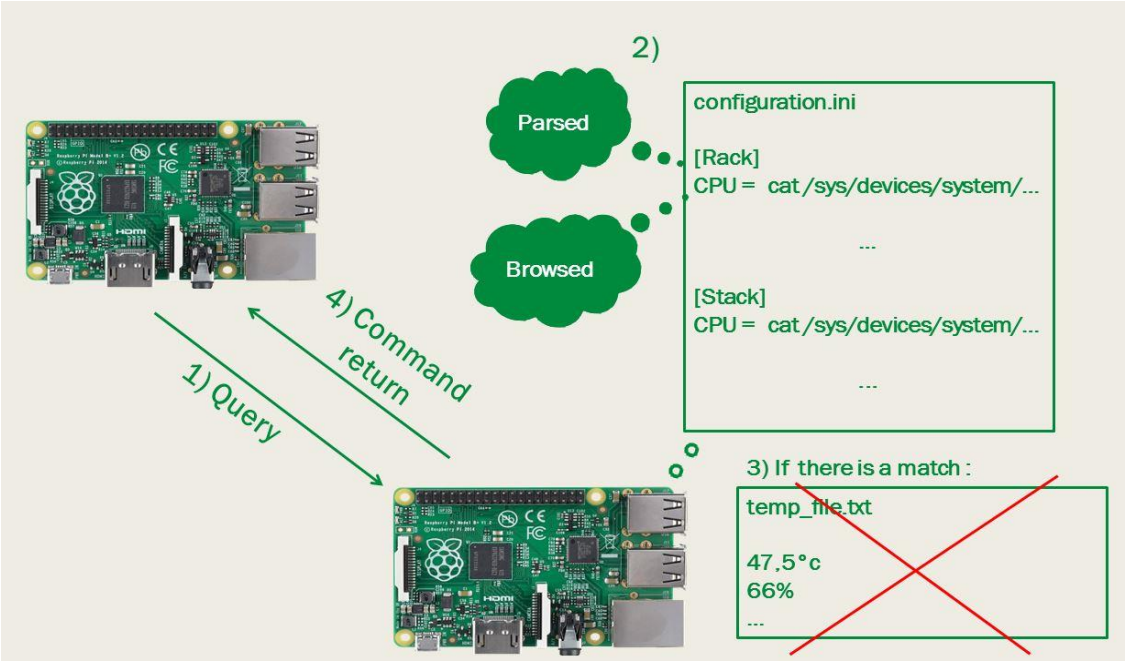

Figure 7 - Command execution process

4 Conclusion

The PiRack developed using a master-slave model was only tested with four stacks of one Raspberry Pi and two stacks of two Raspberry Pis. The forty-nine Raspberry Pis could not be provided by the customers. The different tests allowed to say that the PiRack is currently fast, scalable and run with an asynchronous mode. The web interface developed is responsive and user-friendly as suitable for every kind of user. Finally, the management of the PiRack is really simplified for the testing of distributed algorithms as it was requested by the Labri for the Dionasys Project.

Eventually, particular thanks are addressed to the following supervisors, Dr Laurent Réveillère and Dr Floréal Morandat as well as the representative of the companies Thales, Atos, Sogeti and Orange for their time and their precious advices.

References

[1]The Raspberry Pi Foundation, https://www.raspberrypi.org/.

[2] Dionasys Project, http://www.dionasys.eu/.

[3] Master/Slave model, https://www-304.ibm.com/support/knowledgecenter/ssw_aix_71/com.ibm.aix.genprogc/master_slave_model.htm/

[4] Raspbian, https://www.raspberrypi.org/downloads/raspbian/

[5] DHCP, https://www.isc.org/downloads/dhcp/

[6] AngularJS Framework, https://angularjs.org/

[7] MVC model, https://en.wikipedia.org/wiki/Model-view-controller/

[8]The HTML DOM Document Object, http://www.w3schools.com/jsref/dom_obj_document.asp/

[9] BootStrap Framework, http://getbootstrap.com/

[10] REST API, http://www.restapitutorial.com/lessons/whatisrest.html/

[11] Framework Tornado, http://www.tornadoweb.org/en/stable/

[12] UDP, https://doc.micrium.com/pages/viewpage.action?pageId=10750603/

[13] TCP, https://tools.ietf.org/html/rfc793/

Appendices - I

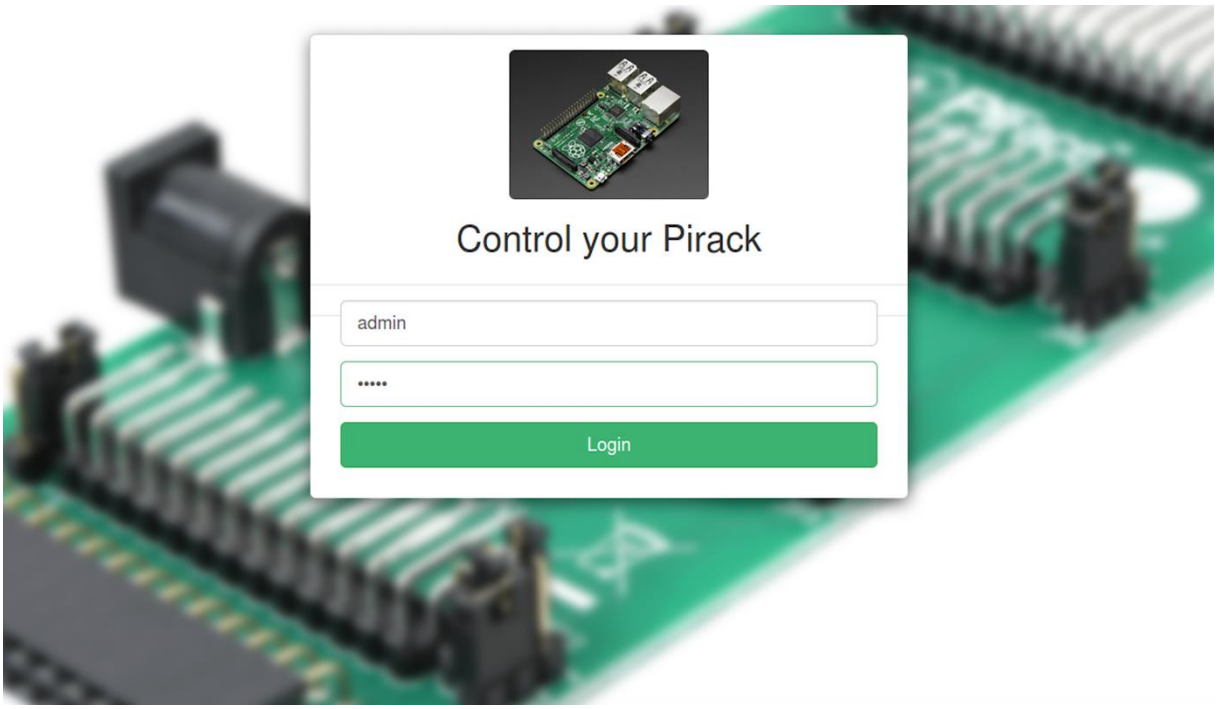

Figure 1: Login page

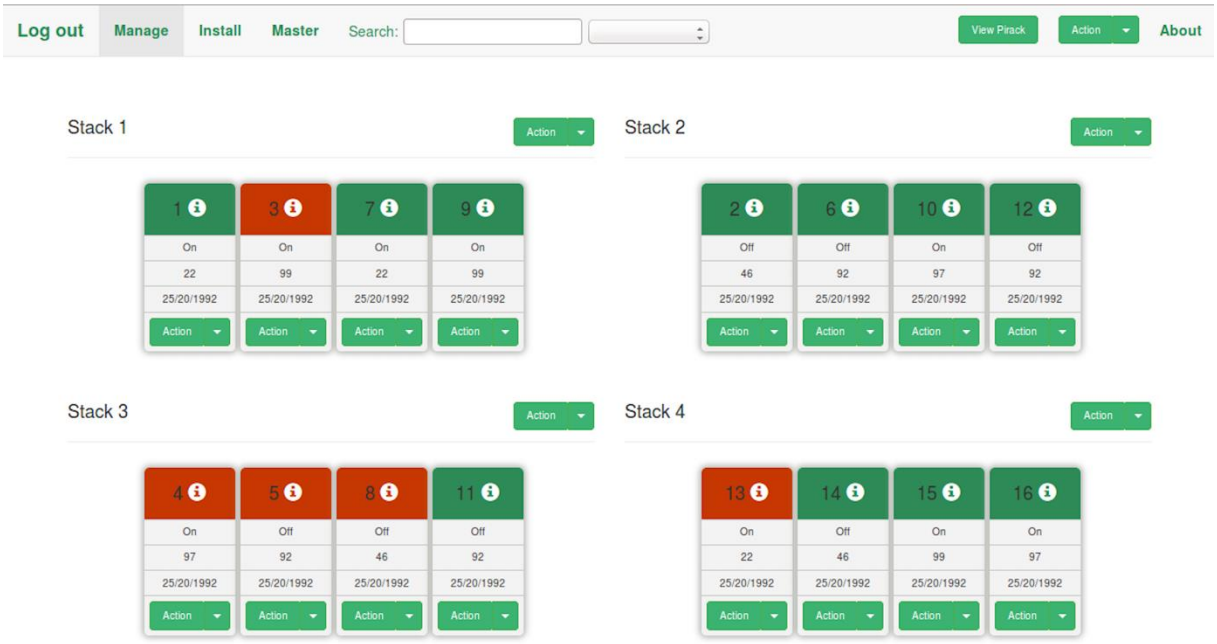

Figure 2: Manage page

Appendices - II

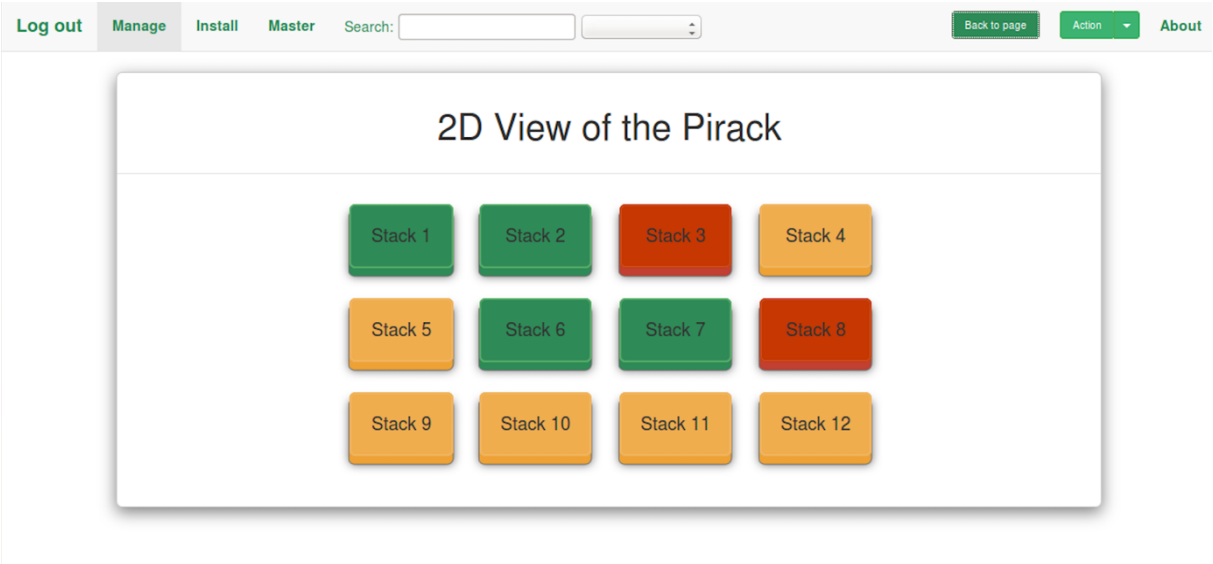

Figure 3: Pirack View

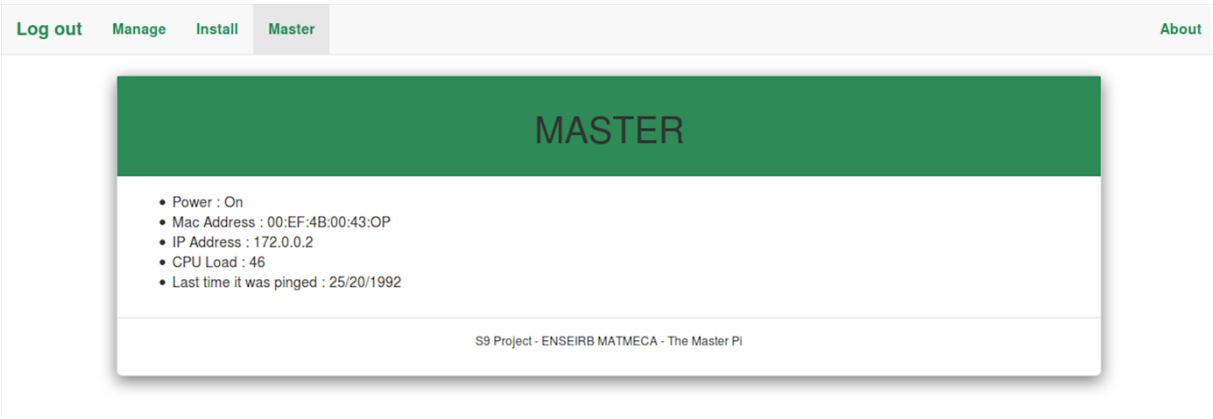

Figure 4: Master Page

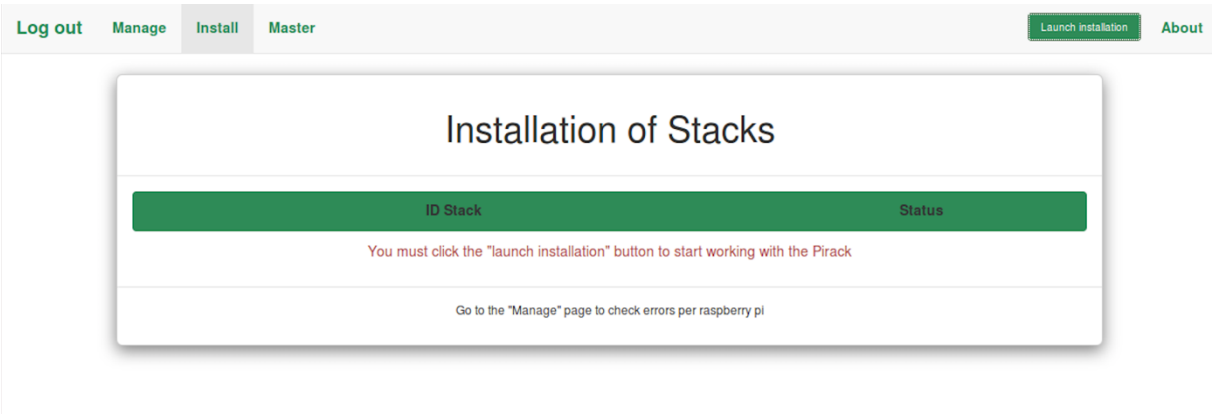

Figure 5: First installation

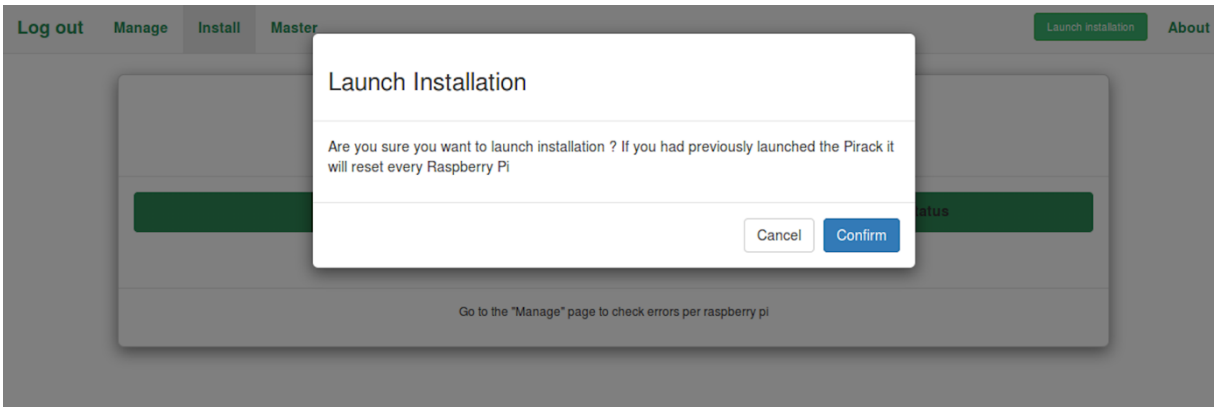

Figure 6: Confirm Installation

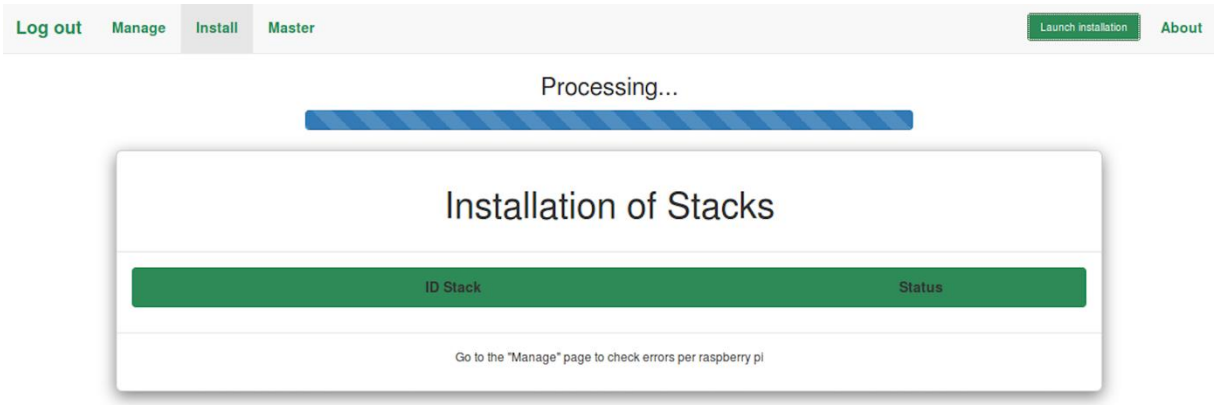

Figure 7: Installation ongoing

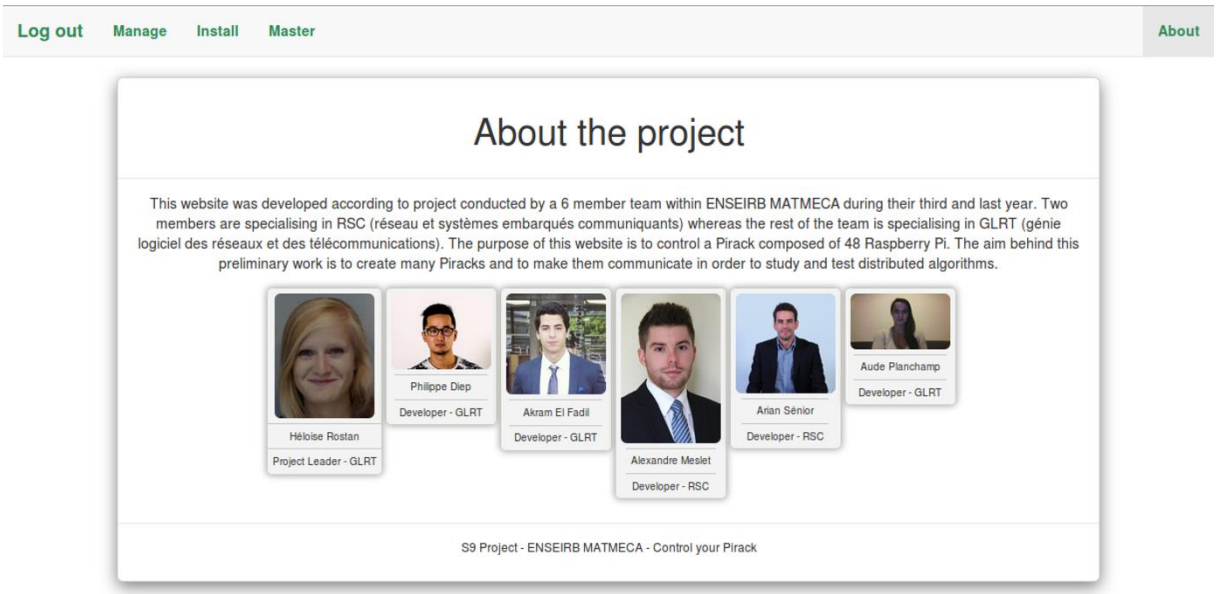

Figure 8: About page