Department of Information Engineering and Computer Science

Master's Degree in
Computer Science

Final Dissertation

Increasing the usability and compatibility of the reCluster sustainable data centre architecture

Supervisor

Lorenzo Angeli

Student

Francesco Bertamini

Academic year 2023/2024

Acknowledgements

I want to thank Federica and Dalí, who are and have been with me at every moment, my friends Aldo and Dennis who supported me and my parents who made it possible for me to get here. I am grateful for my animals, Luna and Gocciolina who are no longer here, but who have always loved me as only they are capable. I also thank my supervisor Lorenzo Angeli for his availability, humanity and professionalism always shown.

Contents

Abstract

ReCluster was conceived with the goal of developing a data center architecture that embodies innovative political, ethical, and technical objectives. Its aim is to provide an eco-friendly, efficient, versatile, and highly flexible solution that allows the use of non-specialized hardware. At the time of this thesis, reCluster was in its early development phase, and the scope for improvement in terms of efficiency, robustness, and usability was still considerable. This thesis undertakes the task of adapting, enhancing, and extending specific features of reCluster to help it transition from a prototype to a production-ready system.

Four main areas of development are addressed throughout this work: expanding hardware compatibility, improving usability, implementing monitoring features, and producing practical user documentation. The thesis presents both the reasoning behind the selection of these areas and the research that underpins the decisions made. It begins by reflecting on reCluster's goals, as outlined in its manifesto, and its potential to provide sustainable IT solutions. Following that, the interventions made are justified through academic literature and other relevant sources. The decisions regarding these interventions are framed by the current state of the project, its declared goals, and the hardware and environmental considerations that guided the development.

For example, the thesis examines which unsupported categories of devices can be integrated into reCluster's ecosystem and why it makes sense to do so, especially in light of typical usage scenarios in academic, corporate, and other environments. It also discusses the usability challenges reCluster initially presented, such as the lack of monitoring tools and comprehensive documentation, and details the steps taken to address these limitations.

A significant portion of the document is dedicated to explaining the technical improvements made, with each change being contextualized, described in detail, and analyzed in terms of how it improves reCluster. After addressing these enhancements, the thesis outlines the remaining limitations, reflecting on how they affect the system's alignment with reCluster's original goals. Additionally, future avenues for further improvements are suggested, providing a roadmap for ongoing development.

The thesis concludes with personal reflections on the project's evolution, assessing how reCluster has grown throughout the development process and suggesting ways in which it could continue to evolve. Technical details of the improvements are provided in the attachments, giving a comprehensive overview of the work carried out to enhance reCluster's effectiveness.

1 Introduction

1.1 What is reCluster

ReCluster is a project that aims to provide clustering software that adheres to specific characteristics, addressing clearly identified and declared problems. From a political perspective, the issues that reCluster seeks to mitigate and resolve are related to the lack of reuse of decommissioned hardware that still possesses useful computing power but is not being utilized.[1] This results in a reduction in the environmental impact of these resources by acting both on the actual reuse of hardware and the extension of the useful life cycle of devices such as computers, which represent a type of waste, the e-wast, that is particularly expensive and complex to dispose of. In the design and construction of reCluster, special attention has also been paid to energy efficiency, enabling the creation of instances that are extremely frugal in terms of energy consumption and resource usage. This characteristic, combined with the fundamental ability to operate with highly heterogeneous hardware, makes reCluster extremely flexible and suitable for use in critical conditions, compensating for the lack of dedicated infrastructures and machines. The first application field for this system is precisely that of self-hosting services within universities.

From a technical perspective, reCluster utilizes and combines various external software components, including Kubernetes, PostgreSQL, NodeJS, GraphQL, and Prisma. It also includes its own components that allow the management of the cluster according to the criteria specified by the project itself. One of the requirements is to minimize dependencies on other software to keep the system overall lightweight and compact.

The operating system on which reCluster is designed to run is Linux, specifically the Alpine Linux distribution. Custom disk images of this distribution are provided, with all the necessary packages pre-installed for execution. Depending on the workload provided, reCluster can scale up or down the number of active and used machines to deliver services. This goal is achieved by automatically turning machines on and off based on the need to have them available for workload distribution.

Since explaining the detailed operation of reCluster is not the focus of this thesis, the functionality of the components affected by the modifications will be explained progressively throughout the exposition.

1.2 Background and Environmental Context

The increasing environmental impact of electronic waste has brought projects like reCluster to the forefront of sustainable IT development. By enabling the reuse and repurposing of dormant electronics, reCluster addresses a growing environmental challenge: the vast

amounts of energy used to build currently unused but still functional devices. Dormant electronics, often stored away in homes and institutions, represent untapped potential, both in terms of computational capacity and environmental burden.

1.2.1 Environmental Impact of Dormant Electronics

Dormant electronics, such as old laptops, desktops, and smartphones, contain significant embodied energy. Embodied energy refers to the total energy used during a device's production, including material extraction, manufacturing, and transportation. On average, the embodied energy of a laptop is estimated to be between 150-300 kWh [2]. However, if these devices remain unused, this energy becomes wasted, as the potential computational power and the resources invested in their production go unutilized.

The environmental impact is not limited to energy consumption; there is also the emission of pollutants during production and disposal. For instance, a typical laptop's lifecycle results in 330 kg of CO2 emissions, of which a significant portion is due to manufacturing [2].

1.2.2 Lifecycle Assessment and Circular Economy Metrics

Assessing the environmental impact of electronics requires various methods. Life Cycle Assessment (LCA) is a comprehensive approach that evaluates the environmental footprint throughout a device's lifecycle, from production to disposal. The circular economy perspective emphasizes extending device lifespans to minimize new production demands. Research shows that extending the life of a device by just one year can reduce the need for new resources by up to 30% [3]. Projects like reCluster play a critical role in this context by ensuring that dormant devices are reintroduced into productive use, thereby decreasing the need for new production.

1.2.3 Conference Insights from Electronic Ecologies

The Electronic Ecologies conferences, held in Australia, have highlighted the urgent need to address the ecological consequences of technological advancement. A core theme from the 2019 conference was the importance of fostering ecosystems that support the reuse and repurposing of dormant electronics [4]. These discussions are highly relevant to reCluster's goal of creating sustainable computing clusters using existing hardware, particularly in academic and business environments.

1.2.4 Reuse vs. Lifespan Extension

The terms reuse and lifespan extension are often used interchangeably, but they reflect distinct approaches to electronic sustainability. Reuse refers to repurposing a device without necessarily modifying or improving its condition. An example of reuse is BOINC (Berkeley Open Infrastructure for Network Computing), which uses idle computing power from personal devices to contribute to scientific research [5].

On the other hand, lifespan extension focuses on maintaining and improving a device's performance over time. By extending a device's usability, as seen in reCluster's architecture, the environmental footprint of electronics is reduced, and the need for new devices is diminished. Research suggests that lifespan extension has a more significant impact on reducing e-waste than simple reuse [3].

1.2.5 Comparing reCluster to Similar Projects

Several other projects have similar goals to reCluster, each focusing on different aspects of sustainability in IT:

- **FUSS**: A project that repurposes old hardware for educational use by installing open-source software on devices in South Tyrol's schools [6]. Both FUSS and reCluster share the goal of reducing e-waste, although reCluster's application is broader, covering academic, business, and personal uses.
- **Biomodd**: An artistic project that repurposes electronic waste into functional art installations [7]. While artistic, the project highlights the potential for unused electronics to be reimagined for new purposes, much like reCluster repurposes old hardware for computational tasks.
- **Datacenterlight.ch**: A Swiss initiative that decentralizes data centers to improve energy efficiency, also reusing server hardware and their open infrastructure. Both Datacenterlight and reCluster focus on minimizing energy consumption and maximizing the use of existing resources [8].
- **Informatici Senza Frontiere**: An Italian organization that promotes the reuse of computers and technology to reduce the digital divide in developing countries and marginalized communities [9]. Both this project and reCluster share the goal of making technology accessible and sustainable by extending the life of hardware.
- **Computing Whitin Limits**: A research initiative that explores sustainable and efficient ways to use computing resources, focusing on the environmental and social impact of computing technologies [10]. Both projects share the broader mission of sustainable computing by optimizing resource use and minimizing waste.

Figure 1.1: Logos of some of the projects that are analog to reCluster.

1.2.6 reCluster's Role in Sustainable IT

ReCluster's impact goes beyond simply reducing e-waste. By facilitating the reuse of dormant hardware, it offers a solution to the rising environmental costs associated with electronics manufacturing. In conjunction with projects like FUSS and Datacenterlight.ch, reCluster contributes to the larger movement toward sustainable IT practices. Its integration of older hardware into productive clusters reduces the need for new device

production and helps extend the useful life of existing devices, aligning with the goals of circular economy metrics [3].

1.2.7 Conclusion

In conclusion, reCluster's efforts to integrate dormant hardware into efficient computational systems address significant environmental and sustainability challenges. By extending device lifespans, reducing e-waste, and optimizing energy consumption, reCluster exemplifies the type of project needed to shift the IT industry toward more sustainable practices. Future iterations of the project could continue to explore the integration of mobile devices and improve usability, further aligning with the goals discussed in this section and following its manifesto [1].

1.3 Interventions' Justification Based On Literature

The work performed on reCluster across the four primary areas of hardware compatibility extension, usability enhancement, system monitoring, and documentation represents a necessary evolution of cluster-based computing systems, grounded in a clear understanding of environmental concerns and emerging technological trends. These interventions can be contextualized and justified by examining relevant literature, which underscores the need for improvements in these areas to enhance reCluster's functionality and environmental contribution.

1.3.1 Hardware Compatibility and E-Waste Reduction

One of the core issues addressed in reCluster is extending hardware compatibility, allowing for the inclusion of a broader range of devices, such as older and less powerful computers. This approach aligns with strategies aimed at reducing e-waste by repurposing existing hardware instead of contributing to the growing electronic waste problem. The ACM paper "The Wild Wild Waste" highlights that e-waste has been a significant environmental concern for over two decades, with vast quantities of unused or outdated electronics contributing to landfill buildup [11]. The issue has only worsened with the rapid pace of technological obsolescence, which drives unnecessary hardware disposal. More recently, the United Nations University reported a staggering 53.6 million metric tons of e-waste generated worldwide in 2019 alone. [12]

Projects like reCluster, which seek to extend the life of older hardware through compatibility improvements, are vital in addressing this issue. By incorporating machines that would otherwise be discarded, reCluster contributes to reducing the environmental footprint of computing infrastructure. The work of Sharma et al. in "E-Inventory for Proactive e-Waste Management" describes a system for cataloging unused hardware in India's public sector to streamline its reuse or recycling, further emphasizing the importance of maintaining functional hardware to mitigate environmental impacts [13]. The extension of reCluster's hardware compatibility directly supports such goals by allowing older devices to continue operating in productive capacities.

1.3.2 Usability Enhancements and Energy Efficiency

The usability improvements made to reCluster focus on reducing the complexity of setup and management, making the system more accessible to a broader range of users, particularly those in academia and small organizations. Simplifying cluster management through enhanced interfaces and streamlined processes also ties into the broader goal of reducing energy consumption in computing systems. As noted by Ricciardi et al. in "Evaluating Energy Savings in WoL-Enabled Networks of PCs," optimizing power management through systems like Wake on LAN (WoL) can significantly reduce energy usage by allowing devices to remain off when not in use and only powering them on when necessary [14]. The integration of WoL in reCluster aligns with this energy-saving approach, as it allows for more efficient power usage without compromising system availability.

Moreover, the need for systems like reCluster to provide greater usability and energy efficiency is also discussed in the paper "The Internet of Tomorrow Must Sleep More and Grow Old," which advocates for systems that can minimize their energy usage during

idle periods while maintaining long-term functionality [15]. By enhancing usability, reCluster ensures that users are more likely to adopt and efficiently manage the system, leading to indirect energy savings through better control over system operations and power management.

1.3.3 System Monitoring: Real-Time Insights and Energy Tracking

The development of reCluster's monitoring capabilities was driven by the need for real-time insights into system performance and energy consumption. This aligns with a growing trend in energy-aware computing, as emphasized by numerous studies. For instance, Ricciardi et al. also discuss the importance of monitoring systems in ensuring that energy-saving strategies are effectively implemented [14]. Real-time monitoring tools allow system administrators to make informed decisions about workload distribution and power management, leading to significant energy savings in networked environments.

Furthermore, the ability to monitor energy usage is essential for addressing broader environmental goals, as discussed in "Technology Replacement and Avoiding Obsolescence: Is It Possible?" by Baker et al., which highlights the potential for repurposed hardware to contribute to energy-efficient computing environments, provided that its energy consumption is properly tracked and managed [16]. The integration of such monitoring capabilities in reCluster not only enhances system management but also ensures that environmental objectives, such as reducing energy consumption and extending hardware lifespans, are met.

1.3.4 Documentation and User Guidance

One of the final areas of improvement in reCluster is the development of comprehensive documentation, aimed at guiding users through the setup and operation of the system. The importance of such documentation cannot be understated, particularly in the context of complex systems like cluster computing. Clear and accessible documentation is crucial for usability, as it ensures users can effectively interact with complex systems. As highlighted by Yadav et al. [17], the presence of detailed documentation significantly enhances software usability by improving users' ability to understand and operate the software, reducing errors and increasing efficiency. Without proper guidance, users may struggle to fully utilize the capabilities of a system, leading to inefficiencies and increased resource consumption.

1.3.5 Conclusion: reCluster in the Context of E-Waste and Energy Efficiency

In summary, the enhancements made to reCluster are not only technically significant but are also supported by broader trends in environmental sustainability and computing efficiency. By extending hardware compatibility, improving usability, implementing real-time monitoring, and providing comprehensive documentation, reCluster is well-positioned to contribute meaningfully to reducing the environmental impact of computing systems. These interventions are supported by a wealth of literature, which highlights the importance of extending hardware lifecycles, improving energy efficiency, and reducing e-waste through innovative system design and management. As computing

systems continue to evolve, projects like reCluster will play an increasingly important role in ensuring that these systems are both efficient and environmentally responsible.

2 Intervention Areas

As anticipated, the thesis work consisted of implementing a series of modifications related to various aspects and functionalities of reCluster and they can be sorted in four areas. In this chapter, I will illustrate which characteristics of the project were affected by practical interventions, providing an overall view of the work carried out for each area.

2.1 Hardware Compatibility

2.1.1 Extending Compatibility with Older Devices

The first modification aimed to introduce support for a specific class of devices that overlaps with another class already supported. These are computers, both desktop and notebook, equipped with processors produced before a certain date. Therefore, the defining characteristic of this class of devices is precisely their production date. Among the machines belonging to this specific class, some will certainly support Wake On Lan technology, thus falling into the already fully supported class of devices compatible with this technology.

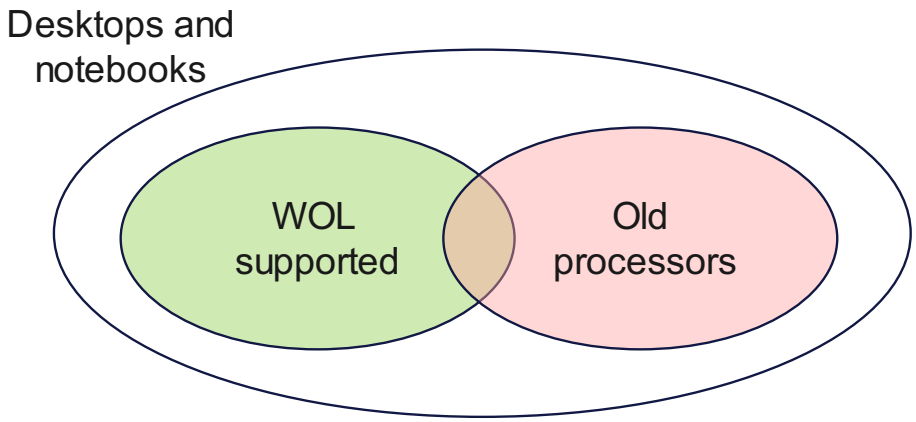


Figure 2.1: The Venn diagram representing the inital devices classes support

2.1.2 Principle of Extending Compatibility to Devices Without Wake On Lan

It was necessary to revise the criteria by which the first version of reCluster did not allow the use of any device that lacked support for Wake On Lan technology. This was, in fact, an essential and exclusive condition for completing the installation process on a node. This condition was imposed even in cases where it made no sense, severely compromising the possibility of using hardware without this support in roles where it was not required. With a view to expanding support for a greater number of device classes, this limitation proved to be detrimental.

2.1.3 Completion of Support for Devices Without Wake On Lan and Addition of Compatibility with Other Device Classes

The most complex and significant intervention of the entire thesis work focused on completing the removal of the limitation related to Wake On Lan technology support, along with introducing compatibility with other device classes. The result, as will be detailed later, led to the introduction of a series of alternative methods for powering on nodes. Consequently, support was achieved for any desktop or notebook computer equipped with at least one network interface. Therefore, this was not just about adding the set of computers without Wake On Lan to the set of compatible ones but about including an entire superclass. As we will see, this superclass specifically includes two additional classes of computers:

- Any computer that supports automatic power-on when electrical power is supplied.
- Any notebook computer equipped with a power button.

2.2 Usability

2.2.1 Use of Graphical Elements

A series of additions and modifications affecting most of the interventions carried out to add interactive graphical components to various elements of the software have been made. The purpose of this choice was to introduce a greater degree of user interaction, in order to provide the user with more control. This results in increased usability, transforming reCluster into software that is easier to use, where the user can be informed and provide input actively and knowledgeably. The introduction of a graphical interface has profoundly changed the nature of the software, making it more accessible and easier to use, even for those who are not expert users of reCluster.

2.2.2 Introduction of Decision-Making Processes

The availability of an interactive graphical interface during the configuration procedures of reCluster allowed for a radical modification of the execution flow. This way, it became possible to involve the user, allowing them to make choices. Consequently, the functionality of various components has been expanded, ensuring that the user is always properly informed, as well as capturing their attention at crucial moments when they are required to make a decision.

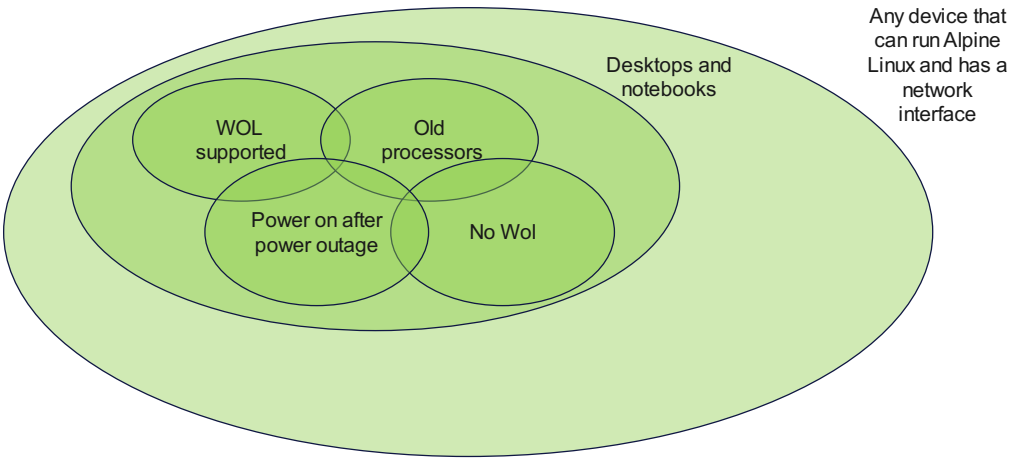


Figure 2.2: The Venn diagram representing the devices classes support after my interventions

2.3 Monitoring

The software has been enhanced in terms of system monitoring, addressing the lack of any tools providing this functionality. Before this modification, the user had no simple way to easily grasp the status and useful information about the nodes making up the cluster. Simplicity and immediacy of use were two key requirements in the implementation. The result is capable of displaying all necessary information, constantly updated. Remote accessibility is also guaranteed, making the proposed solution more usable and effective.

2.4 Documentation

The production of practical documentation has greatly contributed to improving the usability of reCluster. A practical and concrete tool has been provided, in the form of a user manual, which the user can rely on to obtain a working instance of reCluster. Therefore, I have acted to improve the practical use of the software and its components. The intervention in this area allows for better dissemination, facilitating future operators and overcoming difficulties related to the lack of initial knowledge. This enables both efficiency and the collection of a significant pool of feedback, which is crucial for directing future development, fully reflecting reCluster's goals.

2.5 Work Plan

While performing early testing, I discovered that there were certain conditions under which devices produced before a certain date were not supported by reCluster, and the installation of the software on them could not be completed successfully. I developed

a solution that enabled complete support for this type of device. The result was a significant expansion of support, allowing the set of compatible device classes to grow.

In conjunction with other changes aimed in the same direction, I re-implemented the network interface management logic, removing this limitation in light of the imminent addition of alternatives. It was completely removed in cases where it was unnecessary and, indeed, nonsensical. The introduction of this modification represented a significant step forward, although not the complete solution, toward supporting the class that includes all desktop and notebook computers that do not support Wake On Lan technology.

In implementing the graphical interface, I used specific packages that allow the creation of dialog windows and various types of boxes. I added these components wherever the software modifications required them, and wherever it was necessary to capture the user's attention, allowing them to input information and express preferences, assisting interaction with the appropriate level of information.

Regarding the modification of decision-making flows, I evaluated and selected points where operator intervention was needed, as well as where I introduced new branches in the processes and added new functionalities and software components. All these changes were made to support the new features or improve existing ones.

The monitoring tool was implemented using a web interface. This choice proved to be the perfect solution for meeting the proposed requirements. It was developed with a focus on minimizing design, graphical, and implementation complexity. The use of external dependencies and the number of technologies employed during development were kept to a minimum, ensuring broad compatibility and an extremely lightweight solution.

In creating the documentation that serves as a user manual for the current version of reCluster, I proceeded by setting up a cluster from the start. I paid close attention to avoiding assumptions and aimed to cover every procedure as thoroughly as possible, always comparing the content with practical use. I tried to put myself in the shoes of a user entirely unfamiliar with reCluster.

3 Results analysis and discussion

This section presents the results of the thesis work, analyzing their impact, actual contribution, and how they have modified reCluster according to the set objectives. Only the general and relevant technical and engineering aspects necessary for understanding the type of results in question, as well as their consequences on the entire project, will be discussed. All implementation details will be documented in the appendix of the thesis. I will begin by exposing the results in chronological order, according to how I discovered and approached the issues and limitations they aim to address. I will also explain and justify the choices and reasoning that led to the design of the modifications, which resulted in the achievements themselves, in order to maintain a clear logical thread behind the entire work, with the aim of fully meeting the identified and set goals.

The chapter is divided according to the four areas of intervention on reCluster. In sections 3.1 through 3.4, the results of the interventions aimed at extending hardware compatibility are presented. Section 3.1 explains how compatibility with older machines was extended. Sections 3.2 through 3.4 describe the extension of compatibility with additional device classes that lack Wake On Lan support (new network interface management in section 3.2, implementation of new remote node power-on strategies in section 3.3, and development of the button press device for remote power button actuation in section 3.4). Sections 3.5 through 3.8 illustrate the implementation of improvements aimed at usability enhancement through interfaces (the use of graphical interfaces in the setup, section 3.5), automation of procedures (automatic SSH certificate installation, section 3.7, and integrated removal of old installations, section 3.6), and the addition of usage options (basic parameter specification through interface, section 3.8), all aimed at improving usability. Section 3.9 explains how a tool was implemented for system monitoring. Finally, section 3.10 presents the documentation created to facilitate the practical use of reCluster.

3.1 Sysbench Support

One of the essential and characteristic features of reCluster is its focus on efficiency. Therefore, fundamental information includes data on the power consumption of the different nodes that make up the cluster. This data is recorded during the installation process of a new node using a smart plug. The power consumption values of the node under various load conditions are considered to obtain a complete and realistic profile. Measurements are repeated over a period during which the machine is placed under the correct test conditions to ensure proper testing conditions. To profile each node during the installation procedure, ReCluster uses a software called Sysbench. Sysbench is a software that gathers information about a computer's hardware components such as

CPU, memory, and storage devices, and allows to test their performance. As a piece of Free Software, Sysbench's source code is available to everyone, and it is also distributed as a binary in the `community` repository of Alpine Linux.

While testing the installation script, I encountered a problem with some machines that were older than others. The script is structured to test in advance the presence and correct functioning of the commands necessary to complete the installation process. The script would stop due to Sysbench malfunction.

Trying to run Sysbench separately from the command line, it was present but exited with an `Illegal instruction` error.

3.1.1 Context

The cause of this malfunction is the precompiled binary that is downloaded and installed using the packet manager. It was compiled with directives that included optimizations specific to the current architecture at the time of compilation. Since the compilation was done on recent machines used by the repository maintainers, the error occurs on older machines.

On Machine Aanalysis and Comparison

The resolution process began by analyzing the error code with which Sysbench was terminating. From the code, it was evident that the executable was indeed trying to use some instructions that were not available on the current CPU model. On another more recent machine at my disposal, the version of Sysbench installed via the package manager did not exhibit any issues. I then compared the CPU specifications of the two machines, specifically the flags present in the system description files. It emerged that the machine without the problems supported some instruction sets that were not valid for the processor of the older machine. The CPUID level index, which represents the development level of the instruction sets supported by a processor, was also lower on the machine that exhibited the problem. In table 1 are shown the two CPU models involved in the analysis. It can be seen that there is a substantial difference between the launch dates, which justifies the differences in the types of supported instruction sets.

CPU Type	**Model and Family**	**Launch Date**	**CPUID Level**
Problematic	Intel i7-720QM, Clarksfield	Q3 2009	11
Non-problematic	Intel i3-7100, Kaby Lake	Q1 2017	22

Table 3.1: Comparison of Problematic and Non-problematic CPUs

Online Problem Tracing Research

Investigating further, I came across a post on the Sysbench GitHub repository, related to an issue presenting the same problems I encountered. In the reported case, a precompiled executable installed with the system package manager was also used, while the processor in question was an Intel Xeon E5645. Looking at the specifications, it belongs to the Westmere EP family, with a launch date of Q1 2010, not far from the i7-720QM that presented the issue in my case. Checking the specifications regarding the support of instruction sets, I noticed that this CPU had the same CPUID level and lacked to support the same instruction set as the problematic machine I tested.

My research then moved to the issues of the "community" repository of Alpine Linux. Here too, I found a similar report involving an Intel Core 2 Duo P8800 CPU installed on a Mac Mini Server. The family is Penryn, and the launch date is Q2 2009. The same instruction set deficiencies were confirmed in this case as well.

The Alpine Linux Repository

To better reconstruct the origin of the problem, I analyzed the building file present in the Alpine repository, which contains information on how the software was compiled, specifically analyzing the prepare and build sections.

I discovered that a specific flag that is used to tell the gcc compiler not to apply optimizations strictly related to the architecture on which the compilation is performed. The goal is precisely to avoid generating code that uses specific instruction sets and technologies, making the compiled product compatible with the widest possible number of processors, even older ones.

3.1.2 Result

The next step was to find an effective solution for managing the issue where it occurs. A fundamental requirement is to implement an integrated and automated solution as much as possible. The solution to the precompiled Sysbench limited compatibility represents the result of this part of work.

Compiling from source

As suggested by the reconstructive analysis of the origin of the problem conducted, the way to obtain a working binary is to compile from source code on the target machine. This solution was suggested also y the author of Sysbench himelf, in response to one of the issue reports. This way, the autoconfiguration process provides the correct settings to the compiler to produce an appropriate and optimized binary.

However, it is appropriate to perform the onerous compilation process only when necessary, intercepting the different error codes. I also deemed it appropriate to include a certain degree of interaction, involving the user in the decision-making process.

Solution's Implementation

I then structured and designed the decision flow and events involving the management of Sysbench. The Sysbench command is called, specifying a parameter that minimizes the execution time. Its exit code is also intercepted. By analyzing the latter, the user is informed of the situation. If the command is not installed, the user can choose to proceed with the installation via the package manager. Once the operation is completed, the functionality check is re-executed. If there is no internet connection, the user is directly given the option to continue with source compilation.

Otherwise, if the specific error related to the incompatibility of the preinstalled binary with the CPU is detected, the user is informed. A check on the actual processor specifications is then performed to confirm the nature of the problem to the user. This check is done looking at the system files. Whether this specific error or other unknown errors are present, the user is then given the option to proceed with the source compilation of Sysbench to attempt to resolve the problem. All the known uncompatibility conditions are checked, and the outcome of the investigation is shown to the user.

If the user decides to proceed with source compilation, all the necessary operations to obtain a working binary are going to be performed. First, confirmation is requested from the user via a dialog about their intention to proceed. It is also reminded that if there is no working instance of Sysbench, and if they refuse to proceed with the compilation, the entire installation of Sysbench will terminate.

The present version of Sysbench coming from the package manager is then uninstalled, to avoid interfering. A check for the presence of an internet connection is also performed. The next step is to install all the packages necessary to ensure the success of the compilation process. This process is automated and is executed for each package in the list. At the end, the user is informed about the outcome of the operations.

Once the presence of all necessary packages is ensured, a temporary folder is created to contain the products of the various compilation stages. The archive containing the Sysbench source code is then extracted. Then all the generaton and configuration operation are executed automatically. The process is pre-configured to adopt the optimal configuring and compiling options for the purpose (for example avoiding the MySQL support for Sysbench, that in our case is useless). Finally, the make command is executed to produce the binary file.

Once the executable file is obtained, the user is asked whether they prefer to permanently install the compiled version of Sysbench in the system or use it only during the installation process and then discard it. In the first case, the binary file is copied to the `~/bin/` directory. The binary is also renamed as `recluster_sysbench` to distinguish it from the original binary file. Otherwise, it is left in the temporary directory, which will be deleted at the end of the Sysbench installation process. The user is then informed, and the variable containing the Sysbench binary file location is updated accordingly based on the decision made. Finally, the execution check on the Sysbench binary is performed again, to ensure that the executable file produced by the compilation process is working properly.

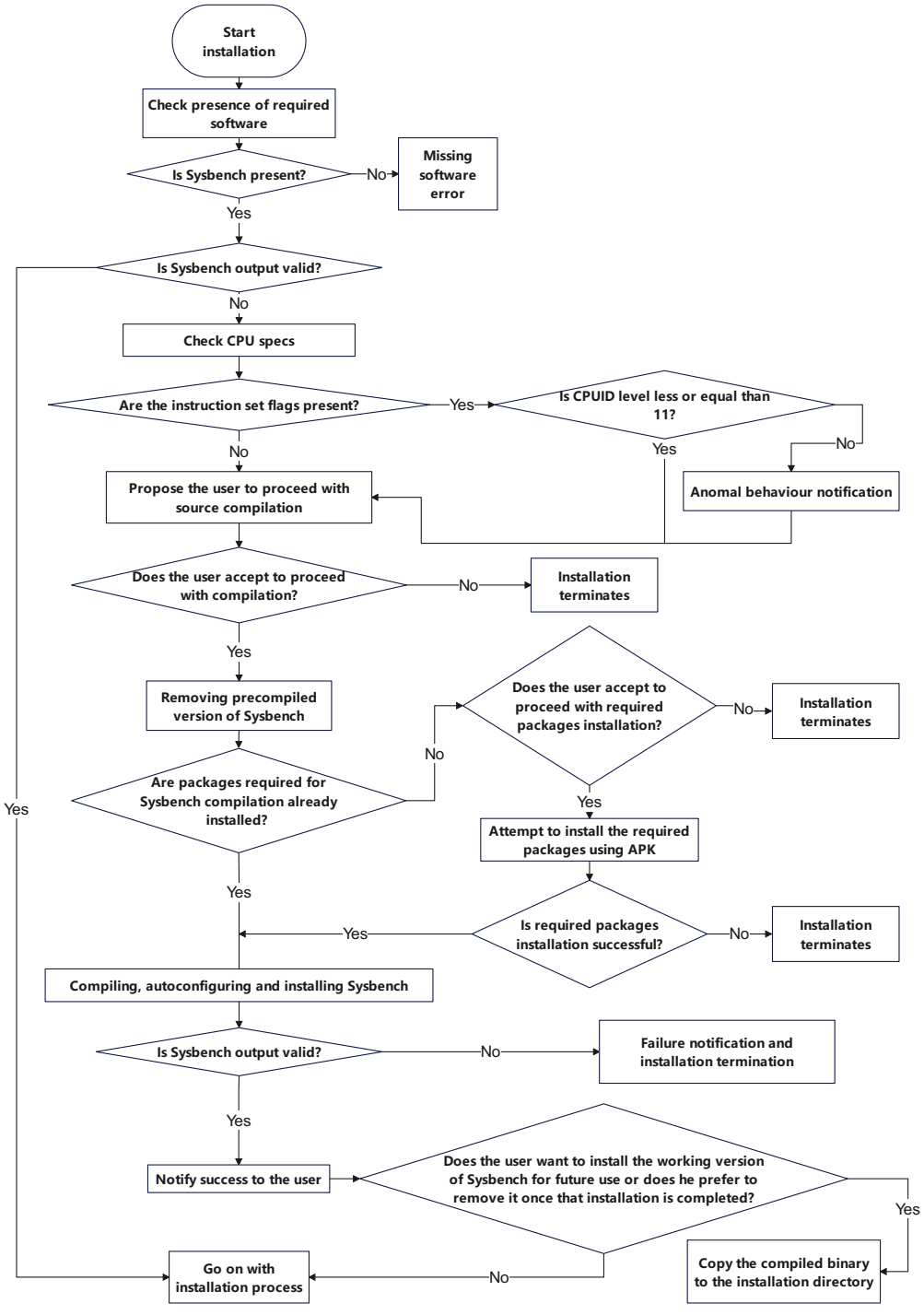


Figure 3.1: Sysbench management flow in reCluster

3.1.3 Discussion

Technical Implications

The outcome of the aforementioned work provides a highly automated and effective solution for addressing specific issues with Sysbench support by the processor in use. This significantly reduces the likelihood of installation process failures. It does so by addressing a compatibility problem with one of the fundamental components necessary to complete the installation process. Several solutions are offered, along with fallback plans that maximize the chances of success.

Additionally, another very important component, previously completely absent from reCluster and its modules, has been introduced: dialogs. I will discuss this in more detail in the dedicated section, but it is important to note how the introduction of a graphical interface, however rudimentary, allows for controlling the behavior of any process, fundamentally changing its nature, including from a technical perspective.

The logical structure of the setup process has been profoundly altered, losing its exclusive sequentiality in the face of any error. A real error handling mechanism has been created to address potential issues that arise from concrete testing activities, following the classification and decomposition of the fault.

Reflections Considering reCluster's objectives, this intervention is highly significant on multiple fronts. The most important result concerns the extension of hardware compatibility. Indeed, all machines equipped with CPUs preceding the introduction of certain instruction sets could not be used with the software "as-is." Functionality could only be achieved by first investigating the causes of the error and then manually compiling from source. This is a time-consuming operation that would greatly compromise reCluster's usability and, consequently, its effectiveness.

There are estimates that suggest an average of at least 5 years during which a consumer computer can provide adequate performance for current software. Considering that reCluster is designed to minimize resource usage and external dependencies, even older computers, potentially affected by Sysbench compatibility issues, can be considered suitable for use. Solving this problem has thus clearly contributed to the effort of removing limitations in supporting as many classes of devices as possible.

Moreover, another crucial component previously absent from reCluster and its modules has been introduced: interactivity. The user is actively informed and has control over the choices that lead to the completion of the setup. This increases user control over the software, allowing them to intervene when a problem arises. This marks the first step in moving away from a "black box" approach, where the only available trace for troubleshooting is long, complicated logs that are difficult and cumbersome to navigate. The level of notifications, as well as interactivity, has been adjusted to provide the most appropriate amount of information to the user without requiring excessive attention. As a result, the setup process remains streamlined, practical, and effective, leading to a notable improvement in usability. Additionally, several procedures (such as compilation), which would otherwise require specific knowledge and significant time, have been successfully automated, sparing the operator from these efforts.

Update on the Alpine Linux Repository

Upon reviewing the Sysbench Gitlab repository for Alpine Linux in June 2024, I noticed a recent modification. The APKBUILD file has been updated, and specifically, the –without-gcc-arch option has been added, ensuring maximum compatibility at the cost of forgoing architecture-specific optimization. Thus, it is likely that the problem was brought to the attention of the repository maintainers, who then resolved it.

However, since I am not running the latest version (3.20) of Alpine Linux but 3.19, the .apk package in the corresponding repository is not updated, unlike the one available in the 3.20 version repository, which is not affected by the problem as it has already been recompiled with the new directives.

3.2 Network Interfaces Management

3.2.1 Context

Staying within the scope of the installation process, another important modification was made, the necessity of which also emerged from practical testing of reCluster compared to the stated goals of its manifesto.

Since this is an architecture aimed at building a data center, communication between nodes, as well as their management and workload distribution, takes place over the network. Therefore, the presence of at least one network interface is essential. Another important architectural factor to consider is that the only tool and method provided for remotely powering on a worker node from the controller was Wake On Lan (WOL). When the controller needed to power on a node to make it available to accept workloads, it sent a WOL packet addressed to the MAC address of the worker machine, thus powering it on.

3.2.2 Result

A crucial phase of the installation process is the scanning of the network interfaces present on the system. In the first release of the software, this process was fully automated and unsupervised. The command `ip` was used to gather information on all available interfaces. All interfaces that did not have, or had disabled, Wake On Lan support were automatically discarded. Interfaces for which no valid uplink speed value was available were also discarded. If there was no interface that met both requirements, the entire setup failed. I proceeded by modifying the entire interface management process.

Solution's Implementation

The process of reading the information of the various interfaces is sequential, as one interface is processed at a time. For each interface, the user is informed about its status and characteristics and is asked about certain configuration choices. At the end of each interface's analysis, if it is deemed suitable for use, it is presented to the user as a potential valid interface to be added to the list.

The first modification consists of adding a check on the activation status of Wake On Lan. It is possible that WOL support is present at both the hardware and BIOS

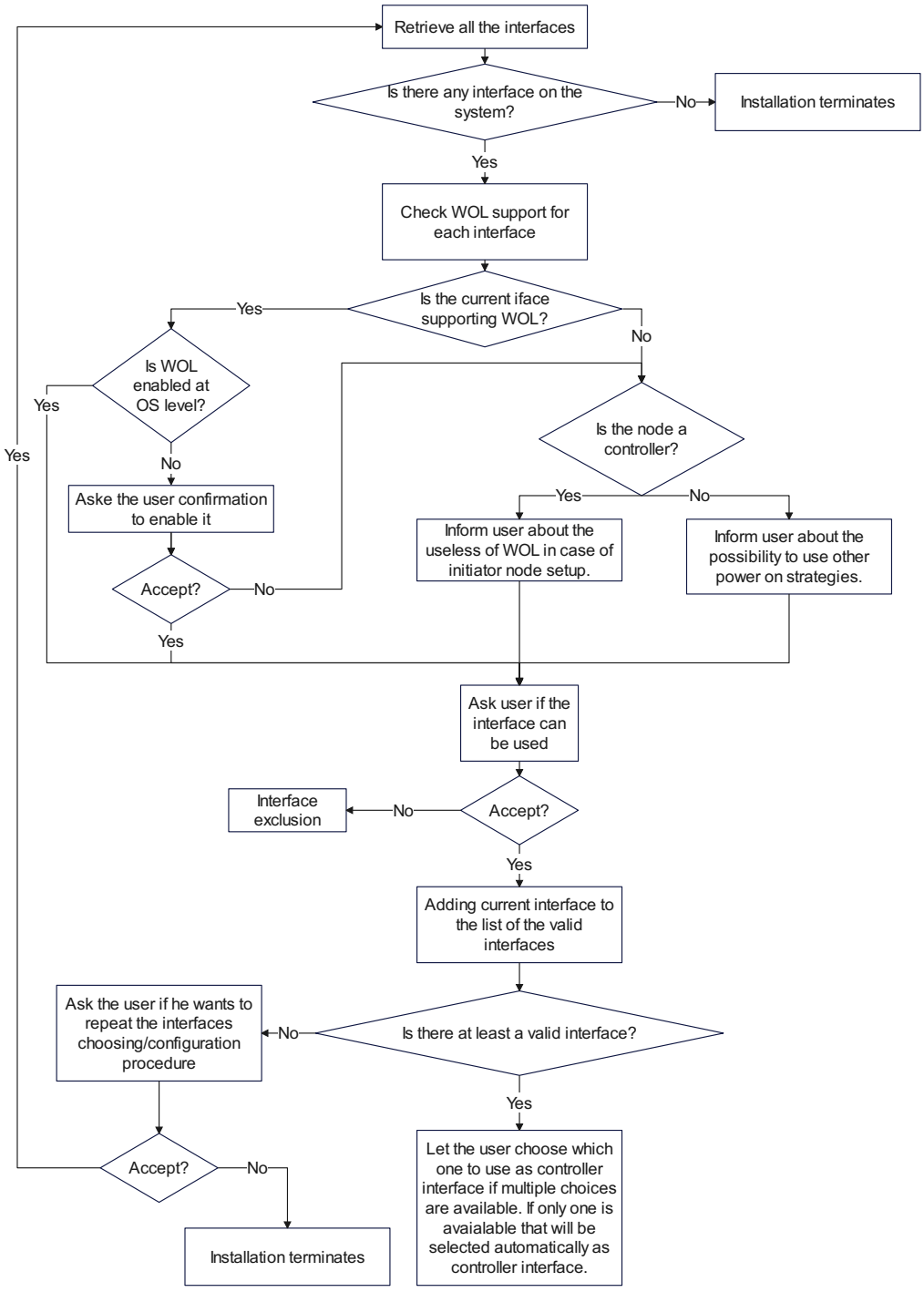


Figure 3.2: Interface management flow in reCluster setup

levels but is disabled at the operating system level. The status of WOL for an interface is indicated by a flag showing its support or status. The flag is a parameter of the interface. If the support is present but disabled, the letter `'d'` is displayed. I made a modification so that if this specific situation is detected, the user is notified via a dialog, from which they can decide whether to attempt to activate Wake On Lan at the operating system level, setting the flag to `'g'`. The user is then informed of the success or failure of the WOL activation. If Wake On Lan is already enabled for the currently analyzed interface, the user is simply notified.

The check for uplink speed remains unchanged. If this value is not valid or is unavailable, it can be assumed that the interface is not connected or otherwise not ready for operation. Therefore, it can be discarded as unusable for any purpose.

At the end of an interface's analysis, as mentioned earlier, its characteristics are summarized. The user can then decide whether to consider that interface as valid or not. The user is also informed based on the role of the machine being set up. Once the analysis of all interfaces is complete, a dialog is displayed listing the interfaces selected by the user, along with their details. The user is then asked to choose one interface to act as the control interface.

At the end of the network interface selection and configuration process, a JSON variable is generated that contains all the information gathered by the system, as well as the user's preferences. This will be passed to the reCluster server component later in the installation process, during the node registration phase. It will be part of a broader variable containing all the information about the node collected during setup.

3.2.3 Discussion

Here I will discuss about the result consequences on the reCluster aims themselves and in relation to the other upgrades that has been done to the whole system.

What The Problem Was

The initial approach presented two problems that highlighted the inadequacy of relying solely on WOL technology, given the project's manifesto objectives. The first issue, immediately evident, is that even during the installation of the controller, the process could not proceed without WOL support. This is contradictory, as the node acting as the controller is the only one that, by definition, remains always on and does not need to be remotely powered on. In a practical example I encountered during preliminary testing, a laptop without WOL could not be used as a controller node. I was therefore forced to discard a valid machine that met all the necessary requirements for that role.

About Device Classes Compatibility

The second, less immediate but equally important, issue concerns the large number of device classes that had to be excluded from use in any role within a cluster. While WOL is a well-established standard in the market, its prevalence, especially among consumer machines, is quite limited, particularly when it comes to BIOS-level support.

The number of potentially useful device classes for reCluster that had to be excluded was substantial. This again significantly limited the effectiveness and fidelity of reCluster to its founding principles.

Interaction and User Control

In resolving this specific issue, a new component of control and interaction for the user was also introduced. The active involvement of the user in the interface selection process brings several positive implications. It is now possible to select a specific interface used for node control, even when multiple interfaces are present, allowing others to be assigned different roles later. Moreover, it is possible to use a different interface for WOL than the one determining the node's IP address. This solution provides greater configuration flexibility. After implementing this feature, the requirement for mandatory WOL support was removed. It is now possible to attempt to enable WOL if it is disabled at the operating system level. The user is also free to consider an interface without WOL as valid, as it will no longer be a limiting factor, since other options for remote power-on will be available.

The addition of interactive management allows the user to be effectively and immediately informed about the device's network configuration, which is one of the fundamental components of reCluster's setup. This facilitates user awareness and grants them greater control during the entire cluster configuration process. In this way, the setup has been prepared for other changes that I will explain in the following sections.

3.3 Additional Power-On Strategies

3.3.1 Context

One of the main issues encountered during the use and deployment of reCluster was the constraint of using Wake On Lan (WOL) as the sole method for powering on nodes from the server. As previously discussed in the introduction, this creates a significant limitation on the range of device classes that reCluster aims to support. The constraints are numerous, both in terms of hardware support and BIOS-level support. Therefore, I decided to address this issue by modifying the individual components of reCluster. Some of the interventions mentioned earlier were preparatory for this type of structural modification to the entire software.

This work led to the addition of three new methods for managing the power-on process of reCluster worker nodes. Some of them involve the use of external and specific devices. Support for these new methods has been added at the database schema, server, and node setup levels. The result is the server's ability to manage and implement the appropriate method based on the information gathered by the installation script during node registration.

The three new strategies are:

1. **Always On:** This involves manually powering on the machine initially and then keeping it always on. It is the least efficient solution in terms of energy efficiency and is designed to be used only when no other alternatives are available.

2. **Smart Plug:** This leverages a feature common to many desktop computers that allows for automatic power-on when power is restored after being disconnected. It involves the use of a smart plug, identical to the one used during the setup phase for reading energy consumption data.

3. **Button Press Device:** By using a custom-designed device, the controller can remotely power on a node by simulating the physical press of the power button.

3.3.2 Result

As previously mentioned, the addition of these features required the redesign and adaptation of three main components of reCluster. Let's look at the details of the modifications.

Database and Entities

The database schema was adapted to include and integrate new entities and attributes necessary for the implementation of the new power-on strategies. The reCluster server's data management system consists of three components that ensure all necessary functionalities. The relational database is MySQL, the ORM interface is Prisma, and the API management is done via GraphQL. In implementing the new worker node power-on techniques, I only modified the configuration of Prisma and GraphQL. As is typical of any ORM system, the structure of the relational database (which in this case corresponds to MySQL database tables) is automatically created based on the directives and schema specified by the ORM itself. Therefore, by modifying Prisma's configuration file, the changes were automatically propagated during the database creation process in MySQL.

This occurs specifically during the execution of the reCluster installation script, in the server configuration phases. Prisma's mechanism is based on generating corresponding migration files when the npx command is executed, using NodeJS's execution tool. Prisma takes the input from the `schema.prisma` file, which contains the database schema, and produces an output SQL file that is executed by the relational database, populating it with the correct tables and fields according to the specified schema. The database address is extracted from the `server.env` configuration file, which is generated by the pre-installation scripts. Typically, the database is located on the same machine that acts as the controller, since the server and database are inseparable components that need to run simultaneously to ensure the cluster's usability.

In production, the typical workflow involves calling GraphQL APIs to execute a query or mutation on the server by a node. GraphQL resolvers rely on Prisma methods and classes to perform type-safe operations on the database. Prisma interacts automatically with the MySQL database, freeing the developer from the tedious task of manually writing and executing SQL queries.

The changes made to the schema are strictly additions, reflecting the maintenance of all previously existing features and functionalities. Specifically, the following changes were made:

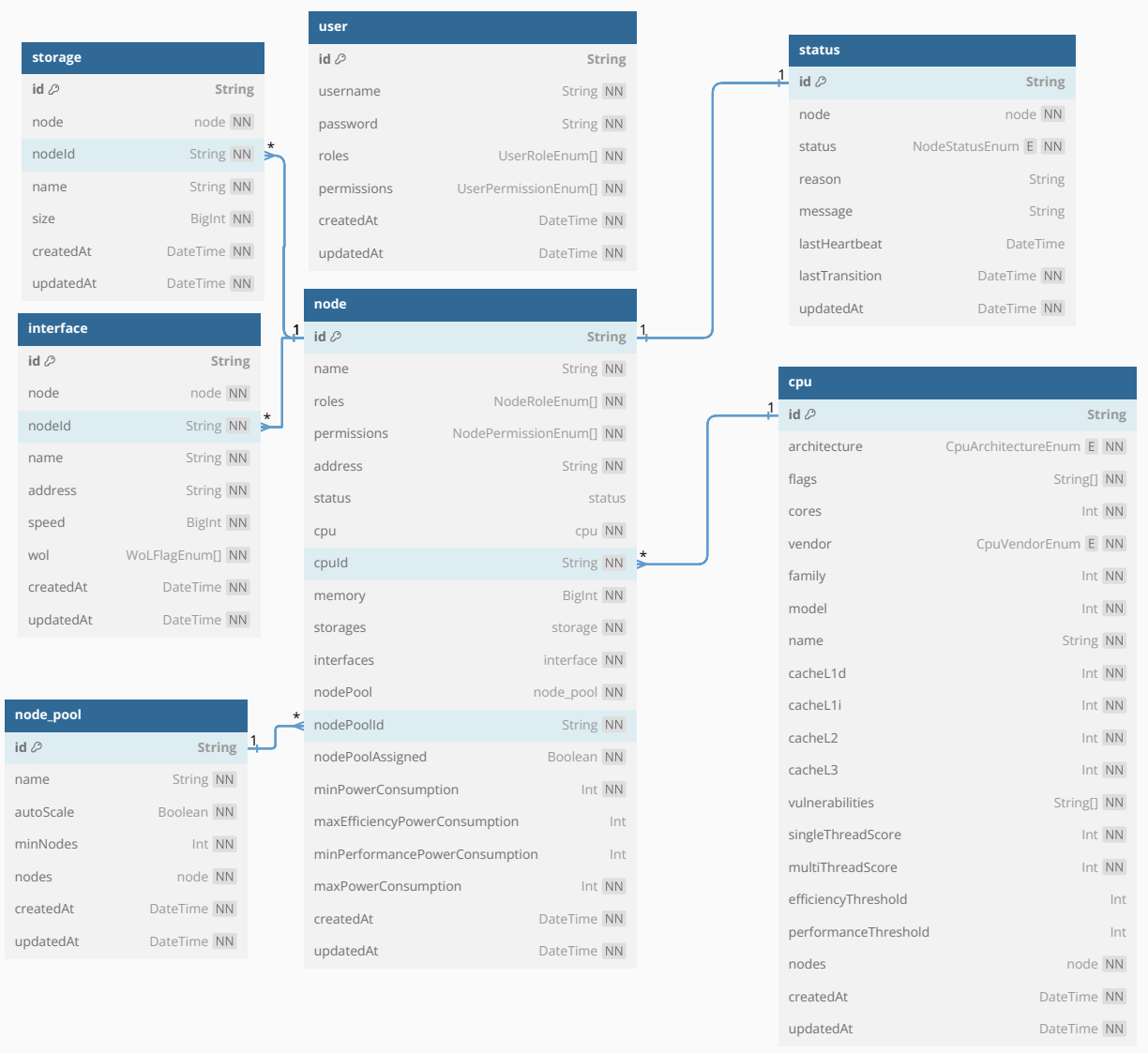


Figure 3.3: The reCluster database schema before the modifications.

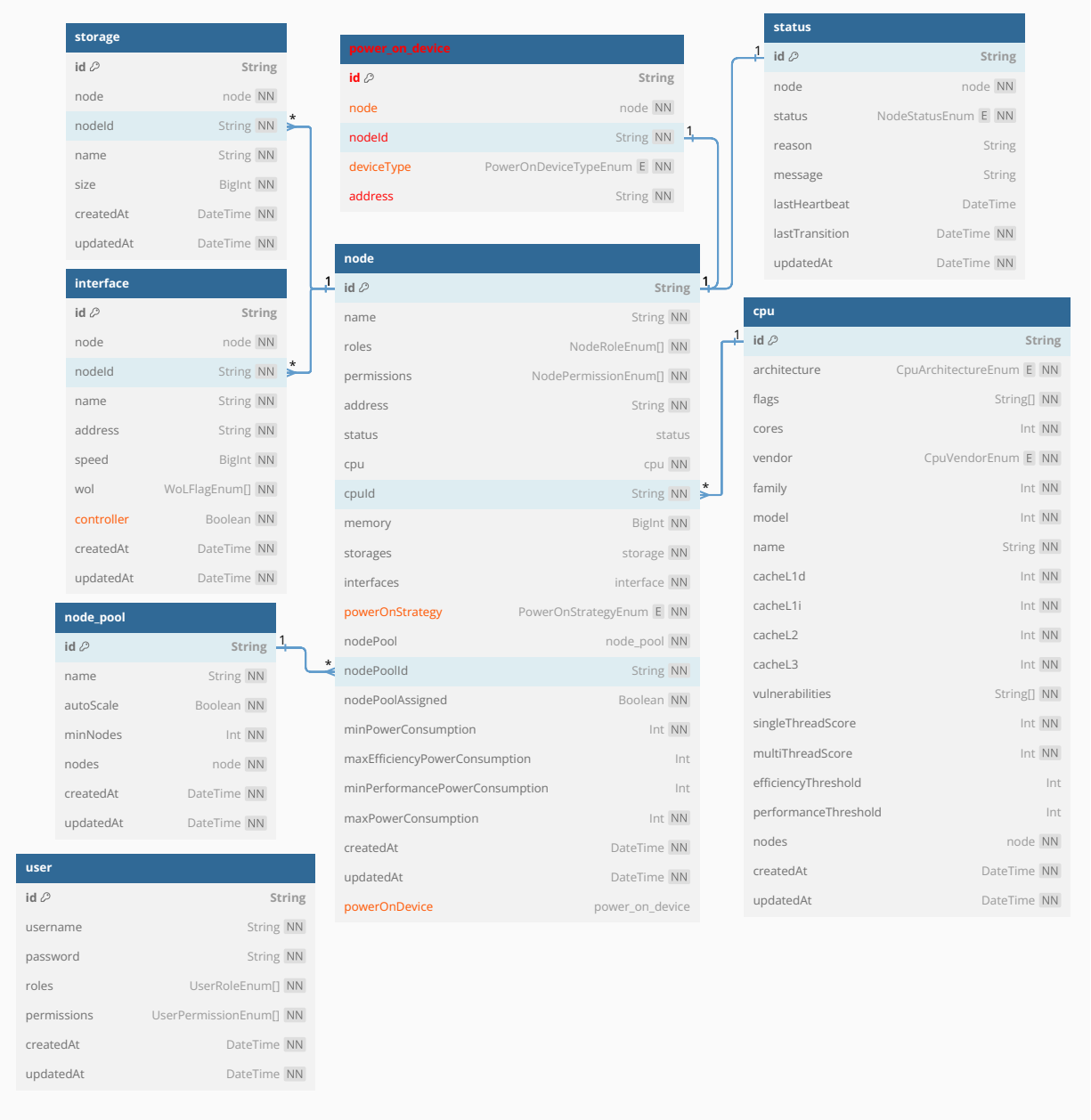


Figure 3.4: The reCluster database schema after the modifications (changes are highlighted in red).

- **Support for the "disabled" state among WOL flags for interfaces:** The possible values for the WOL status flag of an interface now include 'd', representing the case where WOL is supported but disabled at the operating system level on the node.
- **Addition of new power-on strategies:** A field has been added to the "node" entity representing the chosen power-on strategy. The possible values for this field are listed in a dedicated enumerator: "AO" (Always On), "SP" (Smart Plug), "BPD" (Button Press Device), and "WOL" (Wake On Lan). This flag is set during node registration on the server during installation.
- **Addition of support for a power-on device associated with a node:** An optional field has been added to the node entity representing a specific power-on device associated with the node. The corresponding entity contains fields referencing the node to which the device is associated, the IP address of the device, and the device type. This last field is necessary to allow the server to perform the appropriate routines based on the type of device. The possible values are listed in a dedicated enumerator: "SP" (Smart Plug) and "BPD" (Button Press Device).

The next step involved adapting the server code to support these changes in terms of entity properties. This resulted in the modification and addition of files that define and enable the use of methods and objects generated by Prisma, as well as those related to GraphQL API methods. Consequently, the functional server files were updated, as detailed later. This allowed for the implementation of all new procedures by accessing the necessary properties and methods, made available after adjusting the data structure underlying the entire program. New fields were added to class definitions and data access methods (resolvers). New types were also added for use where needed, implementing the new logic for node registration and management on the server.

Installation Script

The setup used for reCluster installation was modified to support the new power-on strategies. Some of the changes mentioned earlier were made to prepare the entire system for extending compatibility with a larger number of device classes using the newly introduced strategies. In implementing these strategies, I started from a version of the installation script that already managed network interfaces correctly. It was possible to proceed without requiring WOL-supported interfaces, regardless of whether the installation was being carried out on a worker or controller node.

The support addition in the setup was developed on two fronts. The first relates to the user interface, used to guide the operator in configuration and collect necessary data. The second deals with packaging the correct data to be sent to the server and implementing the procedures needed to complete node registration.

Regarding user interaction, I created differentiated dialog windows depending on the node's condition. Initially, a check is made regarding whether the control interface supports Wake On Lan. If supported, a single-choice box is displayed with all available

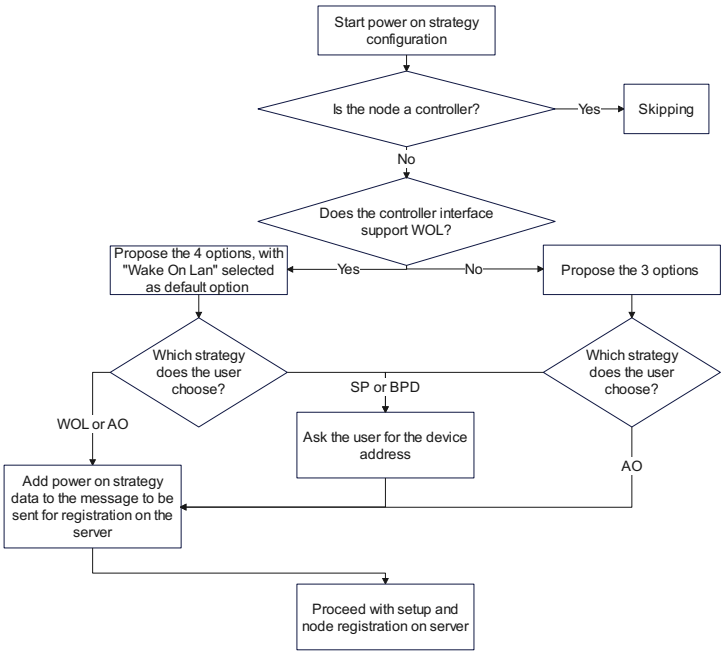


Figure 3.5: The logic flow for the power strategy selection during the setup.

options (four in this case). The pre-selected option is "Wake On Lan" because, if available, it remains the most efficient choice in terms of energy and the fact that it does not require dedicated devices. If the control interface does not support WOL, the "Wake On Lan" option will not be present in the dialog box. In both cases, the situation is clearly explained, and the available options are illustrated. If the selected strategy is "BPD" (Button Press Device) or "SP" (Smart Plug), a box is displayed requesting the user to enter the power-on device's IP address. At the end of the selection process, a summary box displays the preferences expressed.

From a logic standpoint, before starting the power-on strategy selection, a check is performed to determine the type of node being installed. If it is a controller, the entire selection process is skipped, as the server is not governed by any component. Since it must remain always on to ensure cluster operation, there is no need to assign it any power-on strategy, and it is automatically assigned the "AO" strategy, as a strategy must be assigned for registration purposes. If installing a worker node, the procedure proceeds normally. The check is performed on a flag set at the beginning of the installation process. If an IP address for a power-on device is entered, its format is validated. If it is not a valid IP address, the user is prompted to re-enter it. Once the power-on strategy selection process is complete, the collected data (strategy type and, if applicable, the power-on device's IP address) is concatenated in the correct format to the variable containing all node-related data gathered during setup. This variable will then be sent to the server

at registration via a GraphQL mutation.

Server Services

The third and final component updated to support the new power-on methods was the server's node management services. These services handle the operations that power on and off individual nodes. These functions are executed based on directives from the component at the top of the reCluster architecture, particularly the autoscaler. The autoscaler determines which nodes to power on and off based on workload, always maintaining maximum efficiency.

The first release of reCluster was structured to encapsulate all node power-on and off logic within a single service called "NodeService." It contained all the methods for performing actions on nodes, including powering them on and off. There was also a "WakeOnLanService" that implemented the logic for powering on a node via Wake On Lan. My work involved modifying the node service and adding three specific services for supporting power-on devices. These three services are as follows:

- **PowerOnDevice Service:** Contains only the methods related to database operations via Prisma.
- **SmartPlugService:** Contains methods for reading data and controlling the smart plug.
- **ButtonPressDeviceService:** Contains the method for controlling the button press device.

Changes to the "NodeService" were focused on implementing the various power-on strategies, both for powering on and off the nodes, as well as adapting the related data operations methods. The modifications are concentrated on two methods: `boot` and `shutdown`. The first handles the node power-on process, while the second handles their shutdown. Initially, the power-on method only called the Wake On Lan service function to send a WOL packet to the node's MAC address. The shutdown method simply executed a remote shutdown command on the node. Both functions were redesigned to differentiate based on the node's power-on (and consequently power-off) strategy. Let's take a closer look at the changes:

Function `boot`

- **AO Strategy:** No action is taken, as the system assumes the node is already on. If the node is off, it must be powered on manually. It remains on after installation.
- **WOL Strategy:** The WOL support for the node's control interface is checked. The function to broadcast the WOL packet is called, passing the node's control interface MAC address as an argument.
- **SP Strategy:** The presence of a smart plug associated with the node is checked. The smart plug's address is retrieved from the database, and the function to power

it on, located in the smart plug service, is called. The smart plug's functionality is explained in detail later.

- **BPD Strategy:** The presence of a BPD device in the database is checked. Its address is extracted, and the method in the dedicated service is invoked to send the press request to the device.

Function `shutdown`

- **BPD and WOL Strategies:** The `sudo poweroff` command is sent via SSH to the node. This is possible due to the key exchange that took place during the installation process.
- **AO Strategy:** No action is taken, preventing the node from being shut down and keeping it active.
- **SP Strategy:** First, the remote shutdown command is executed via SSH, as with the BPD and WOL strategies. Then, a dedicated method checks whether the node is reachable using the ping command for a specified number of times (up to 2 minutes). After the node is unreachable for 15 seconds, the smart plug is turned off, cutting power to the node. This delay ensures the shutdown has completed, avoiding the risk of hardware or software damage from an improper shutdown.

To complete the picture, it is necessary to analyze the methods in the button press device and smart plug services in detail.

- **ButtonPressDeviceService:**
  - `pressButton`: This function sends the press request to the device at the specified address, making an HTTP request using the REST APIs configured on the device. It also handles errors related to communication issues with the device.
- **SmartPlugService:**
  - `getPowerState`: This function queries the smart plug (whose address is specified as a parameter) for the current power state.
  - `setPowerState`: This function sets the desired power state using the smart plug's API, making an HTTP request. The argument is either "ON" or "OFF" and the plug's IP address.
  - `togglePowerState`: This function is used when the node needs to be powered on. It checks the current state of the plug. If it is "ON" (for example, if it was manually reactivated and not by the reCluster server), it turns it off for 2.5 seconds and then turns it back on. This allows the initially powered node to transition to a non-powered state before powering on automatically when the power is restored.

- `powerOn`: This function powers on the node by calling the `togglePowerState` function, ensuring the correct power sequence.
- `powerOff`: This function simply uses the `setPowerState` method to turn off the plug. The timing of this action (the appropriate moment to execute it), as already illustrated, is ensured by the functions in the `NodeService`.

Thanks to the interaction of these components and the modifications made to them, the four strategies are fully implemented and integrated into various parts of reCluster. Further implementation details are provided in the dedicated section of the appendix.

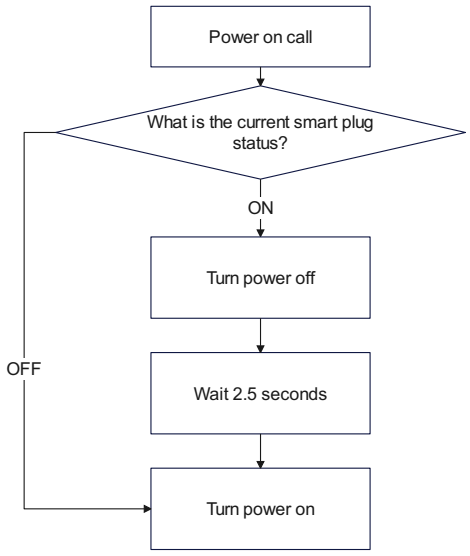


Figure 3.6: The logic flow for the smart plug to power on a node.

3.3.3 Discussion

Extending Compatibility

This intervention was one of the most significant applied to the entire software, given its great relevance to reCluster's ability to adhere to the principles and goals upon which it is based. Reflecting on the scope of extending compatibility with other device classes, the importance of introducing the three new power-on strategies becomes evident. They allow the use of virtually any desktop or laptop within a cluster. The "Always On" strategy also introduces the ability to support any device on which reCluster can be installed. Obviously, its use is not recommended, as this strategy undermines reCluster's energy efficiency goals, regardless of the machine's type. However, it offers a crucial tool that allows reCluster to maintain maximum effectiveness even under particularly unfavorable conditions, where suitable hardware or power-on devices are unavailable.

The reCluster architecture ensures that priority is always given to available worker nodes (those already powered on) and those that are most energy-efficient. This minimizes the negative impact of adopting this strategy, reducing idle time for nodes that are always on.

It is also important to note that the newly supported device classes are undoubtedly the most common in academic organizations' inventories. This fact supports the increased usability in the primary environment where practical use of reCluster was envisioned.

Usability

Usability was also kept at the center of the design process for the implemented interventions. As for the setup, the user is never required to question or remember details about the node type or its characteristics. The user is always guided to make choices that are appropriate and correct based on the current state. All necessary information is provided to the user, ensuring a clear understanding of the situation and context in which a decision is being made. All interactions occur within the same type of graphical interface used throughout the installation script. The same applies to the server modifications, where every event and possible error is logged. Even in the presence of a problem, the reconstruction and resolution process is simplified. The guarantee that nothing incomprehensible or unexplained happens is maintained, ensuring the operator always understands the system's behavior. I also paid great attention to maintaining consistency in terminology so that the user is never confused about the meaning of a term, helping them become familiar with reCluster's specific vocabulary. This leads to greater confidence and assurance during operations.

Working Methodology

During the process of implementing the new power-on methods, I followed an incremental methodology, alternating between implementing a change in one component (such as the data structure) and then propagating it to other components (such as the installation script and server) to maintain constant cohesion within reCluster.

3.4 Button Press Device

3.4.1 Context

As seen in the previous section, one of the new power-on strategies involves introducing a device specifically designed to allow the server to press the power button on a notebook to turn it on. In this section, we will discuss the various design aspects, from hardware and design to firmware and the necessary software tools for its operation.

3.4.2 Design and Hardware

Design

The device consists of three hardware components: a frame on which an ESP8266 microcontroller and a small servo motor are mounted. Once assembled, the three elements form a device with a structure similar to a small clamp. The device is positioned around the notebook chassis, aligning the cam that presses the button with the button itself. By adjusting the locking screws, the two halves of the frame tighten around the notebook, holding the device securely in place.

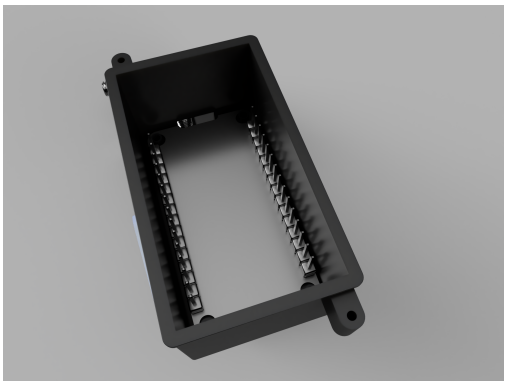
The top part of the device houses the control board. It has a cover with ventilation holes, as well as openings for the cables connecting to the servo motor and the USB port. The cover snaps into place but can also be secured with two screws. Inside, four supports hold and fix the board in position. The height extension has been calculated to comfortably accommodate the connectors for the cables running to the servo motor. The servo housing has been carefully sized to ensure a perfect fit, keeping it firmly in place. However, two front holes have been provided for additional securing with screws.

The two inner surfaces of the frame in contact with the notebook will be covered with a few-millimeter-thick layer of adhesive foam polyurethane. This prevents any damage to the notebook and provides additional grip. The frame length and the grooves for the adjustment screws allow access to buttons located at different distances from the chassis edge. The dual hinge allows for adaptation to notebooks of different thicknesses. The nuts used for securing are of the self-locking type to prevent loosening over time. The device's overall width has been minimized to ensure maximum compatibility with different notebook designs.

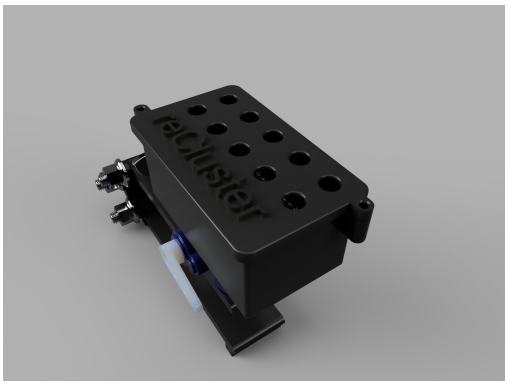
Hardware

The control board is equipped with an ESP8266 chip manufactured by Espressif Systems. This is a widely used chip in the IoT space due to its extremely low cost (70mA) and advanced features like WiFi. It has very small dimensions and low power consumption. The board features a USB port for communication and power. It is possible to power a small servo motor, like the one used in this device, via the dedicated pins that share current from the USB port. The board is equipped with eight analog output pins that can be used to control the servo motor. A pulsed signal (PWM) is transmitted to the pin, determining the servo motor's position based on its frequency. The controller can be programmed in C using the environment and libraries commonly used for other development boards like Arduino.

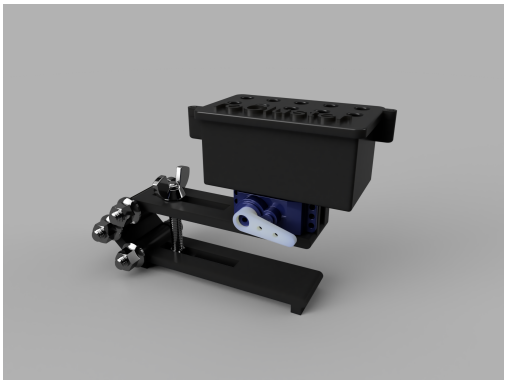
The servo motor used is the SM-S2309S model. It operates at 5V and can thus be powered by the control board, which in turn is powered through the USB port. It consumes 0.44 amps under no load and 1 amp when stalled. Weighing 9 grams, it has a depth of just over 2 cm and a width of just over 1 cm. It has a total rotation angle of 60 degrees and provides up to 1.1 kg*cm of torque when powered at 5 volts. Its casing is made of hard plastic, and the three power cables are about 20 cm long. The market price for a single unit is around 4 €. Given its compactness, low consumption, power,



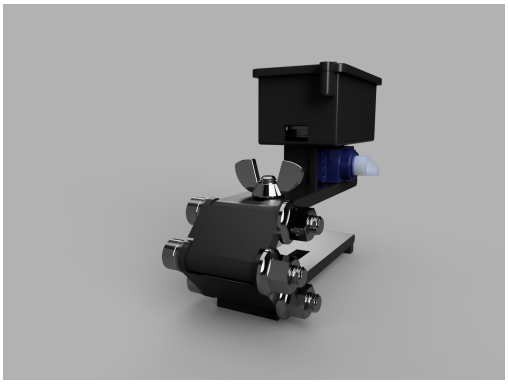
(a) The board container



(b) The board container cover



(c) The frame



(d) The double adjustable hinge

Figure 3.7: Gallery of BPD renders.

Table 3.2: Specifications of the SM-S2309S Servo Motor

Model	SM-S2309S
Operating Voltage	5V DC
Idle Current	0.44 A
Stall Current	1 A
Weight	9 g
Dimensions (LxWxH)	23.8 x 12.3 x 21.6 mm
Torque	1.1 Kg * cm

and affordability, this servo model is perfectly suited for use in the BPD device.

The device frame is made using 3D printing technology with filament. The material used is PLA (polylactic acid), a biodegradable plastic made from starch, aligning with reCluster's eco-sustainability focus.

3.4.3 Firmware

The firmware of the control board is fairly simple. It contains two main functions: the setup function and the loop function. The loop function is continuously executed after the setup function runs once at startup. Let's analyze the content and structure of the firmware.

All parameters are saved in global variables, which are initialized based on the user's settings before the firmware is uploaded to the board during the configuration process. The configuration parameters are:

- **WiFi SSID:** The name of the WiFi network to which the device should connect.
- **WiFi Password:** The password for the WiFi network.
- **IP Address:** The static IP address to be assigned to the device.
- **Rotation Angle:** The angle by which the servo motor should rotate when triggered.
- **Press Duration:** The time, in milliseconds, the servo should stay in the rotated position, corresponding to the duration of the button press.
- **Servo Pin:** The pin to which the servo motor's control cable is connected.

The logic flow is very simple. The first operation after startup is to return the servo to its initial position. Then, the device connects to the WiFi network and retrieves the network configuration parameters. If the WiFi connection fails, a log message is printed to the serial port, and the loop function is executed, iterating the connection attempt. The only network parameter required is the device's IP address; the subnet mask and gateway address are automatically retrieved upon connection.

Once these operations are complete, several endpoints are instantiated to allow the device to be used via API. Specifically: /areyoualive: Simply responds with HTTP 200 and prints "OK". It is used to verify that the device is connected and active. /press: This is the endpoint used by the reCluster server to operate the device and power on the node where it is mounted. The servo motor is triggered according to the rotation angle and press duration values specified in the variables. It responds with HTTP 200 and the message "Success" if the execution (at the software level) is successful. /test: This endpoint allows testing and adjusting the mechanical behavior of the device. The HTTP request specifies two parameters: angle and press time. The servo motor is then triggered based on these parameters, allowing the user to quickly find the appropriate configuration for the notebook where the BPD will be mounted.

All log messages, whether they indicate success or error, are always printed to the serial port. This allows for simple, automated monitoring of the device's behavior (as I

will illustrate shortly), providing all the necessary information for debugging and configuration tools used by the user.

3.4.4 Configuration Script

Along with the development of the device, I also created a shell script for configuration. It is capable of gathering all the necessary parameters, packaging and uploading the firmware, and finally testing the device's functionality. The entire process is built using interactive graphical elements offered by Whiptail, just like the reCluster installation script.

The script uses "arduino-cli" to compile and upload the firmware, which provides the Arduino development environment features via the command line. Let's analyze the structure and operation of this script in detail:

- **Parameter Collection:** The user must specify the settings (WiFi SSID, WiFi password, device IP address, servo rotation angle, press duration, and servo pin) to compile and upload the firmware with the correct values. The script allows these parameters to be specified as command-line arguments. If any parameter is invalid or not provided at runtime, a series of dialogs will be displayed for input. For the servo angle, press duration, and pin, the script suggests default values that are likely suitable for the BPD. The script validates all input values.

- **Library Installation:** After c ollecting the necessary parameters, the script checks for the presence of the libraries required to compile the firmware. If they are missing, the script attempts to install them using "arduino-cli". If a library installation fails, the script exits.

- **Firmware Generation:** The firmware code is embedded in the script. At this point, the variables representing the values in the firmware are replaced with the user-specified values, packaging the final code for uploading to the device.

- **Compilation and Upload:** A check is performed on the serial port to verify the device's presence and connection. If the serial port is not detected, an error message is displayed, and the user can restart the configuration process. If no issues are detected, the script proceeds with compiling and uploading the firmware to the device.

- **Output Monitoring:** Once the firmware upload is complete, the device is rebooted. For the next 60 seconds, the script monitors the device's serial port output, looking for any error messages. If problems occur, the user is informed and advised accordingly. The user has the option to restart the configuration process from the beginning. The case where the serial port is accidentally disconnected is also handled.

- **Testing:** If no problems are detected based on the messages received from the device, the script contacts the device via network at the "/areyoualive" endpoint.

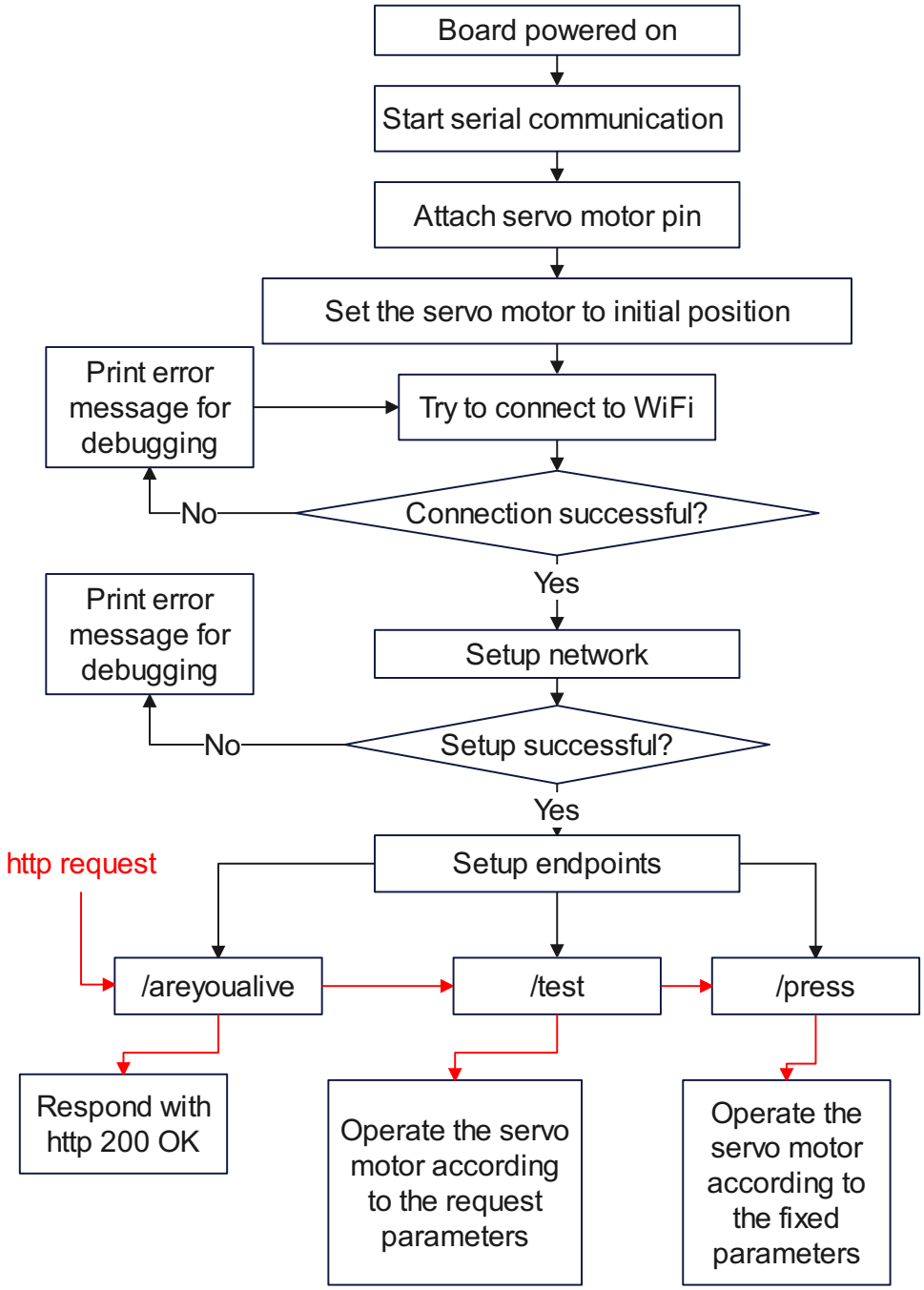


Figure 3.8: Button Press Device firmware logic diagram

If the device is unreachable, appropriate feedback is provided to the user. Otherwise, the script proceeds to test the device's functionality. The user is notified to observe the actual movement of the servo motor, after which the "/press" endpoint is triggered to operate the device. The user is then asked if the device worked correctly. If the response is positive, the script terminates successfully. If negative, the user is advised to check the connections and is offered the option to retest.

To ensure proper script functionality, the following dependencies must be installed: "whiptail" (for the graphical interface), "arduino-cli" (for firmware management), and "curl" (for making requests to the device).

3.4.5 Testing Script

To make adjusting the device easy and fast, I created a simple script that allows for rapid and efficient testing. Once started, the script prompts the user for the IP address of the device to be tested. It then requests the rotation angle and press duration to be applied. It proceeds to send the properly formatted request to the device's "/test" endpoint. After informing the user of the test result, it restarts, retaining the last entered values. This way, it is possible to quickly test various angle and press duration values. The script requires the "whiptail" and "curl" packages to function.

3.4.6 Discussion

The development and design of this device, along with the results obtained, have made a significant contribution toward bringing reCluster closer to its ideal and project definition. Specifically, the addition of a device class that includes notebooks without Wake On Lan or power-on-after-power-loss capabilities is particularly impactful. Given the widespread use of these devices in environments typical of reCluster, such as universities, this improvement has a highly positive impact on the project's overall effectiveness.

The device's characteristics align perfectly with reCluster's core principles. The architecture is simple, consisting of very few, common, and inexpensive components. Its design ensures minimal costs. The configuration tools make it practical and easy for any user to operate, requiring no specific technical expertise. Clear and effective documentation is provided to guide users through all phases, from assembly to installation. The software and design are open-source, encouraging easy replicability and dissemination of the device. The construction and operation are accessible to anyone who needs it. The device can be used on a wide variety of machines with different shapes and sizes without the need for further modifications. It is made from environmentally sustainable materials. The device also has a high repairability index due to its completely modular design. If a component breaks, it can be easily and affordably replaced, reducing unnecessary waste.

As illustrated, this is not just a device but a complete set of tools designed to make its use as intuitive and practical as possible.

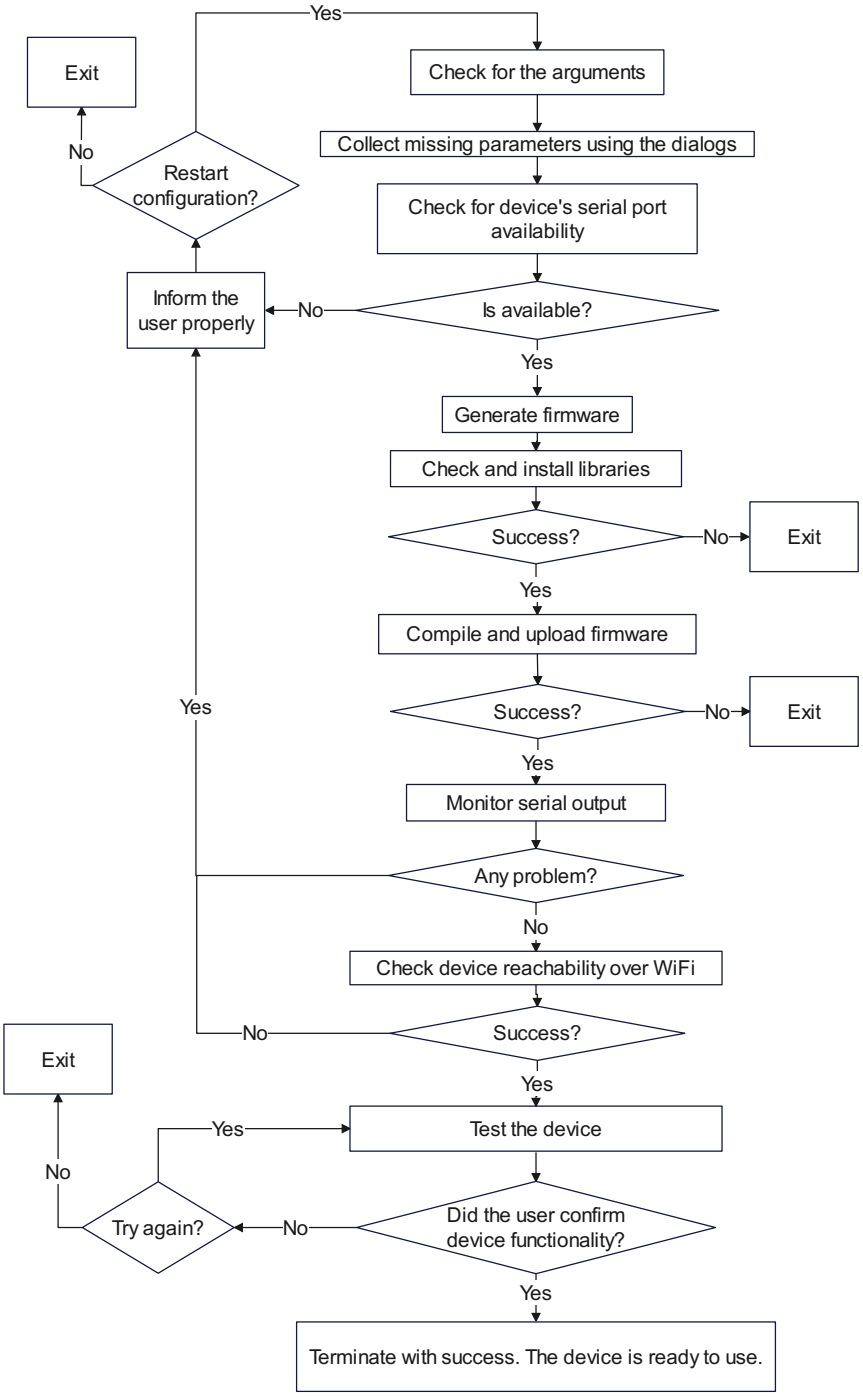


Figure 3.9: Button Press Device configuration script logic diagram

3.5 Interactive Graphical Interface Inside the Setup Process

3.5.1 Context

During the initial approach and testing of the installation process, I immediately noticed the complete lack of any user interaction. The only source of information for the user running the application was the log produced by the script. However, due to its complexity and length, it is difficult to maintain effective real-time control. Even when reviewing the log after the fact, navigating through the long series of generated information can be time-consuming and far from intuitive. In the initial version of reCluster's setup, there was no provision for any choices or active supervision by the operator. The only parameters were specified as script arguments. Following the functional modifications made to both the system as a whole and the setup specifically, I realized the need to address this limitation.

3.5.2 Result

Therefore, I deemed it necessary to introduce a system of interactive dialog windows in order to provide the ability to display instructions, execution information, and settings. To implement these elements, I chose to use Whiptail, a library that allows for the creation of dialogs and informational boxes in the console in a simple and effective manner. It is installed using the APK package manager and is therefore perfectly integrated with the Alpine Linux distribution used by reCluster. Creating the interface elements is very straightforward, involving calls to specific functions where configuration parameters, such as size, type of dialog, options, title, and informational text, are specified. A major advantage is how seamlessly the dialogs integrate into the flow of the script's execution. The script pauses whenever user input is required, and it is easy to direct the execution based on the user's selection when a choice is needed. The types of dialog boxes used include simple informational windows, confirmation dialogs (Yes/No), text input fields, and single-choice dialogs from a list of options, such as radio buttons.

3.5.3 Discussion

The interactive interface elements were introduced at various points in the setup process. As a result, the installation script has been improved in terms of both manageability and usability. The user is prompted and required to intervene only when necessary. I made sure not to overuse graphical elements, as excessive user interaction would reduce usability by distracting from the most important information. The setup has therefore moved away from its initial "black-box" nature and now provides useful information for its completion. Decision-making flows have also been made clear and easy to understand. The user no longer has to guess or make assumptions about what is happening or about the consequences of their choices, as each step is clearly explained, leaving no room for potential misunderstandings. This outcome reduces the technical understanding required by reCluster operators.

Thanks to this transformation, the software's dissemination is also facilitated, as its use now requires less effort, both during installation and troubleshooting. This modifi-

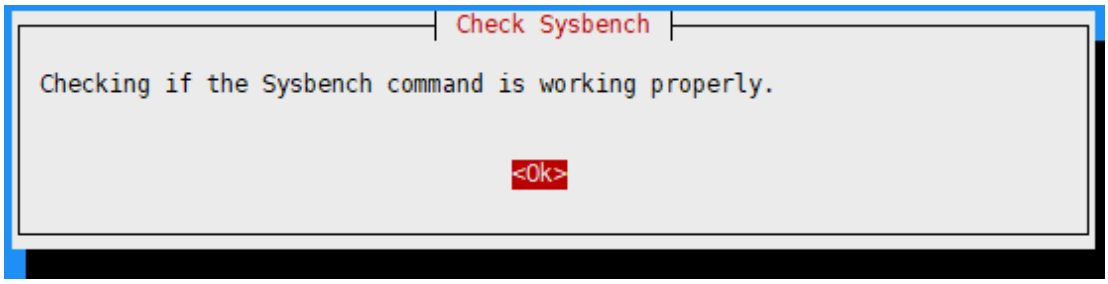


Figure 3.10: An example of an informative dialog made with Whiptail

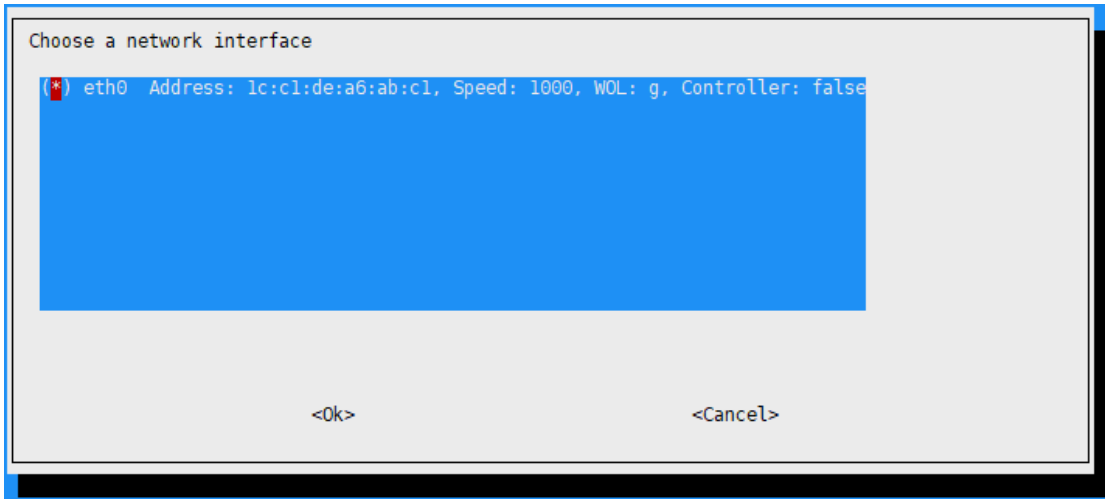


Figure 3.11: An example of a radio selection dialog

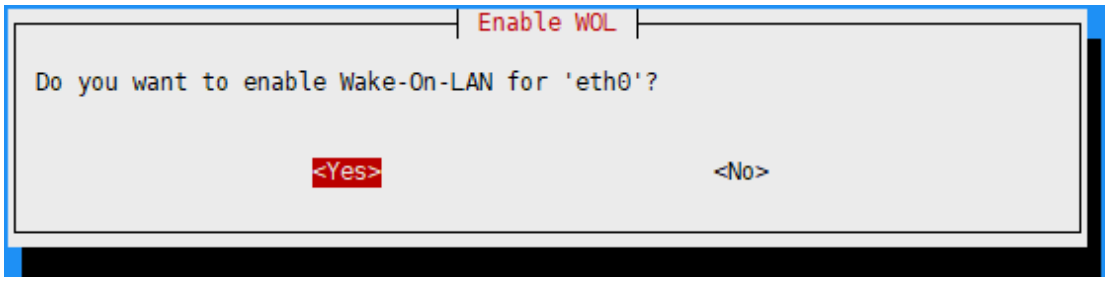


Figure 3.12: An example of confirmation dialog

cation, therefore, contributes to expanding the "human compatibility" of the system by breaking down barriers related to difficulty of use and user awareness. reCluster thus achieves a higher level of maturity, bringing it one step closer to meeting the requirements for production use.

3.6 Automatic Removal of Old Installations

3.6.1 Context

During the numerous initial attempts to set up reCluster, I encountered a relatively annoying recurring event (at least in the early stages of testing). Once an installation was completed or interrupted, when trying to perform a new installation, the setup process would stop, correctly indicating that the reCluster system folders were already present. This behavior is absolutely normal. Depending on the installation's completion level before the interruption or completion, residual components (files, services, etc.) are always present and must be removed before proceeding with a new installation.

To address this, since the first version of reCluster, an uninstallation script was created. It is installed on the target system at the start of the setup process, before any trace of reCluster is left behind. In this way, even if the installation fails at any point, the system cleanup tool will always be available. At the end of its execution, the script deletes itself. I then decided to include the automatic execution of the uninstallation script, when necessary, within the setup process.

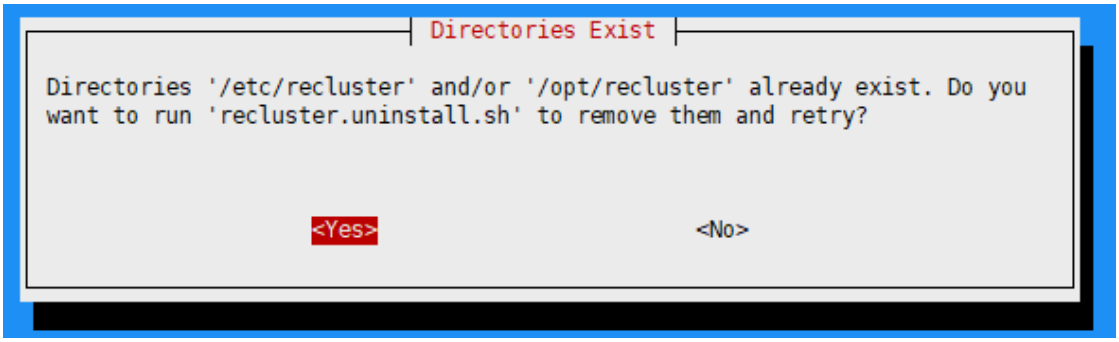


Figure 3.13: The uninstall request dialog

3.6.2 Result

The implementation of the automatic uninstallation mechanism turned out to be quite simple. At the beginning of the script's execution, a check is performed to verify the presence of reCluster files in the system folders. If the check yields a positive result, a choice dialog is presented to the user. The user is notified of the presence of an old reCluster installation and the need to remove it. If the user agrees, the uninstallation script is simply executed, after which the setup process continues as normal. If the user declines, the script proceeds without cleaning the system and fails when the check is performed again.

3.6.3 Discussion

Thanks to this small adjustment, the number of failed installation attempts is reduced, making the process smoother and easing the user's task. By automating this step while still allowing for user supervision, the overall usability is improved, compensating for the typical oversights of operators.

3.7 Automatic Installation of the SSH Certificate

3.7.1 Context

One of the tasks required from the user during the reCluster installation process is the installation of the SSH certificate produced by the server onto the worker nodes. This enables the server to execute remote SSH commands on the worker nodes, such as the shutdown command. The manual installation operation consists of copying the contents of the server's certificate file and adding it to the `~/.ssh/authorized_keys` file on the node. I decided to implement a system that allows the user to choose whether to automatically download and install the certificate during installation.

3.7.2 Result

The implementation involved both the installation script and the server. On the server side, an endpoint was instantiated, from which the setup can download the certificate. This endpoint is located at `http://server_ip:3000/api/getcertificate`. It was implemented within the same web server already created to serve the web dashboard. The endpoint simply serves the `ssh.crt` file located in the `/etc/reCluster/certs` folder on the server. As for the installation script, after completing all other operations and only in the case that the node is not a controller, the user is prompted via a dialog box to choose whether to proceed with automatic installation. The server's IP address is extracted from the configuration file. If the user accepts, the certificate is downloaded and inserted into the appropriate file. If the download fails or the user refuses automatic installation, an informational dialog box appears, notifying the user and advising manual verification of the certificate's presence. In earlier setup phases, the certificate from the installation configuration files had already been copied in advance. Unless the user specifically prepared it, this will contain only a certificate generated by the node itself, making it useless.

3.7.3 Discussion

The addition of this simple automation relieves the user from a repetitive and tedious operation. This eliminates the need for a manual process, which is time-consuming and adds complexity. This impact is particularly noticeable when installing a large number of nodes. Once again, the user is free to act as they deem most appropriate, with a choice. The usability and practicality of the installation process have once again been improved.

3.8 Specification of Basic Parameters via Graphical Interface

3.8.1 Context

The first version of reCluster required that the configuration parameters necessary for installation be specified only as arguments when running the setup shell script. This solution is practical and time-efficient as it does not require any interaction after the execution starts. However, for a less experienced user or someone who hasn't had time to become familiar with the large number of parameters that can be specified in reCluster's configuration, this solution may seem less intuitive. In fact, there is a set of parameters that must be specified for each installed instance and cannot be effectively and coherently initialized with default values. Examples include the IP address of the smart plug used to measure the node's energy consumption and the administrator password. If one of these or some other parameters is not specified or has an invalid value, the installation fails.

Therefore, I decided to modify the behavior of the installation script so that if one or more parameters are not correctly specified, a specific dialog for collecting the missing values is automatically shown to the user. In this way, the user cannot overlook the lack of vital information for the successful completion of the installation process and is guided in providing them. On the other hand, if the entire set of required parameters is correctly specified as script arguments, no dialog is shown. The parameters in question are:

- **Log level:** Although not vital and could be set to a default value, I decided to include it as it is important for the user to consciously choose the level of detail for the information provided by the script.
- **Admin password:** The password for the administrator user on the server.
- **Autoscaler password:** The password for the user within the autoscaler component of reCluster.
- **Smart plug IP address:** The address to reach the smart plug responsible for energy measurements on the node.
- **Configuration file:** The path to the file containing all the parameters necessary for the node configuration.
- **K3S configuration file:** The path to the file containing K3S settings.
- **Cluster initialization flag:** The flag that determines whether the node being installed is a controller node or not.

3.8.2 Result

At the implementation level, I modified the `parse_args` function, which is responsible for reading and interpreting the arguments. I added simple validation functions for

each of the basic parameters, along with auxiliary methods. An example is the method that concatenates the IP address of the smart plug used for measurements, creating the correct URL to access its API. I also added dialogs for each parameter. If the value entered in the dialog is not valid, the user is prompted to re-enter it. I did not modify the parsing of command-line arguments. To trigger the potential appearance of the various dialogs, I initialized the global variables that hold the parameter values as empty. In this way, if the reading of an argument does not result in a value being entered, the variable remains empty. A check is then performed on each variable, and for those that are empty, the corresponding dialog is shown.

3.8.3 Discussion

Once again, it became clear that there is considerable room for improvement in managing decision-making processes and user interaction in this first version of reCluster. The goal, even in this specific case, was to relieve the user from possible pitfalls and errors that may occur during the installation process. The user is thus safeguarded, while still having the option to operate as in the initial version, which is less user-friendly but more practical in certain contexts. Consider the case where the installation must be replicated across many nodes: it is more convenient to specify the parameters as command-line arguments, saving the time required to fill out each dialog.

3.9 Web Dashboard

3.9.1 Context

During practical use of reCluster, I realized how cumbersome and complex it was to retrieve information regarding the system's status. It became clear that there was no element that could provide an at-a-glance description of the cluster's structure and status. In fact, the only way to retrieve information about the nodes was to run commands from the terminal. However, this lacked immediacy and a clear representation of the entire system's status. Therefore, I decided to develop a web control interface that would display all the information needed to provide a complete and updated overview of the system's current status. It displays all the nodes, their status, characteristics, interfaces, and any associated power-on devices.

3.9.2 Result

Client-side

The web dashboard was built using extremely basic web technologies, minimizing the use of non-essential libraries, frameworks, and components. Only pure HTML, Javascript, and CSS were used to create the client-side interface.

The layout choice consists of a series of horizontally extending tables. The entire page is divided into sections, one for each node. Within each section, there are two or three tables. The first table contains the node information:

- ID
- Name

- Status
- IP Address
- Roles
- Power-on strategy

The second table contains interface information, one row per interface. The columns are:

- Interface name
- MAC address
- Speed
- Wake On Lan support flag
- Controller flag (true/false)

Finally, the third table is present only if the node has an associated power-on device (i.e., if it is configured with the SP or BPD power-on strategy). In this case, the fields are:

- IP address of the device
- Device type

The first section is always occupied by the node acting as the controller, which is highlighted in yellow. At the top of the page, the dashboard's update status is displayed. If the last update succeeded within the expected time (i.e., receiving updated data from the server), the status is "OK" (indicated by a green dot). If there were communication problems with the server, the status is "Failed" (indicated by a red dot).

The entire page is dynamically generated and managed by a Javascript script integrated into the HTML code. The script retrieves all necessary data from the "/api/nodes" endpoint exposed by the server. Iterating over the received data, it generates the corresponding DOM elements and applies the necessary formatting. The script also repeats the generation process at a predefined interval. If the request to the server fails and the data "expires," the script highlights the failed update status by modifying the relevant page elements, providing a clear visual indication to the user. The update timeout is 10 seconds.

The page's style is defined through CSS within a dedicated style section. As previously mentioned, no third-party components were used in the implementation, simplifying the architecture by using only pure Javascript, HTML, and CSS.

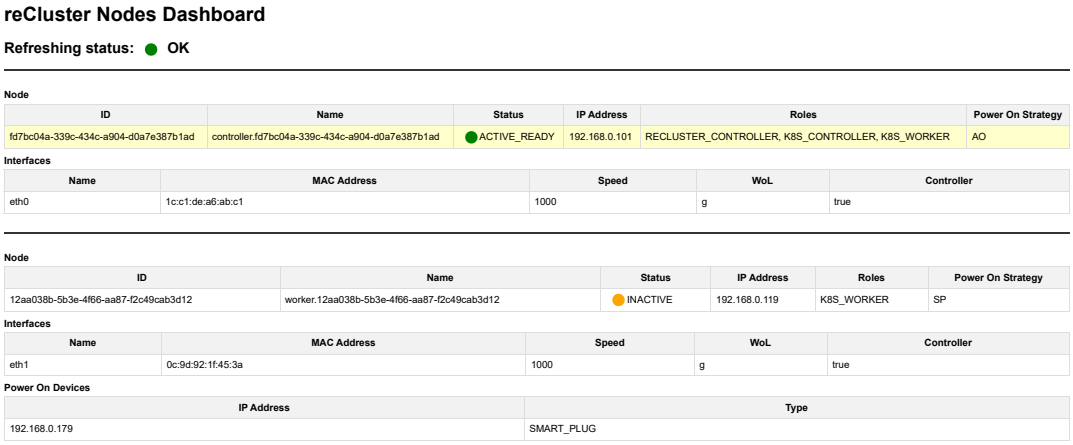


Figure 3.14: The reCluster web dashboard

Server-side

On the server side, the choice of implementation strategy was fairly straightforward. Naturally, I opted to include the dashboard functionality within the reCluster server. Since the server was already developed using the NodeJS framework, I chose the popular "Express" package to implement the web server. The web server listens on TCP port 3000 and exposes two endpoints:

- **/dashboard:** Navigating to this endpoint serves the HTML file containing the dashboard. Entering only the IP address and port 3000 automatically redirects the user.

- **/api/nodes:** This endpoint is used by the Javascript code of the dashboard to retrieve all necessary data in JSON format to dynamically generate the page content.

A controller is also implemented to retrieve all the data served by the "/api/nodes" endpoint from the database using Prisma methods.

3.9.3 Discussion

The implementation of a web dashboard further enhances the usability of reCluster. Users can now rely on a tool that provides immediate access to an always-updated snapshot of the system's status. With this tool, numerous useful details about the cluster structure can be quickly and effectively retrieved.

Once again, the entire web interface was developed according to the core principles of the project. It was implemented by minimizing dependency on external software, resulting in a product with a very high degree of compatibility with virtually all modern web browsers. The server-side implementation is also perfectly integrated with the other

existing components of the reCluster server, ensuring the software remains cohesive and robust.

3.10 Usage Documentation

3.10.1 Context

When I attempted to install reCluster on a node for the first time, the only source of practical information available was the Wiki. It was little more than a draft, and the information it contained was sparse and shallow. I realized the importance of having a user manual for a complex system like reCluster. For a new user, the amount of information to keep in mind is overwhelming, and without a clear guide to follow through the necessary steps, using the system becomes complicated and difficult.

I, therefore, completely restructured the Wiki, producing a practical and easy-to-consult user guide. I designed the structure to ensure that all the steps leading to a fully functional installation of reCluster could be executed effectively, while also facilitating the search for specific information.

3.10.2 Result

The process of creating the documentation from scratch was completed as I performed the installation of a reCluster instance at the university using the new version I had developed. This allowed me to cover all the phases and procedures before, during, and after the installation. The guide contains only practical instructions and descriptions of the necessary action sequences, without theoretical notions. A minimal technical description of the architecture is provided to contextualize the content. As a true user manual, it includes practical examples to ensure the user fully understands the descriptions.

Below is the structure of the documentation:

- **Introduction to the documentation**: The purpose of the documentation is explained, and the flow of the installation process is illustrated, offering an important tool for user orientation.
- **Architecture Overview**: Brief descriptions and diagrams are provided to represent how a cluster is structured at both physical and software levels. The different power-on strategies, a key topic for user understanding, are also illustrated.
- **Requirements**: Hardware and general requirements for the various cluster components are detailed.
- **Preparation**: The user is guided through the procedures to be completed before the actual installation. This includes cluster design, network, and hardware preparation, including accessory devices.
- **Installation**: All installation steps relevant to the user are covered, where decisions need to be made, preferences expressed, and choices taken. Each step's functionality is explained, along with the possible options and their impact on cluster configuration.

- **Testing**: Some testing procedures that can be performed post-installation are explained.
- **Troubleshooting**: Suggestions are provided for checks and solutions to the most common issues and anomalies encountered.

The documentation is structured to be effective when read and executed from start to finish, always following the sequence of steps required for the practical execution of the installation, making it an ideal support. While composing the TiddlyWiki containing the documentation, I took care to organize the sections exactly as described in the index. I included valid content from the existing incomplete documentation to ensure efficiency and provide a comparison with what had been written by those who worked on reCluster before me.

3.10.3 Discussion

Reflecting once again on the goals of the reCluster project, there is a clear alignment between the creation of this documentation and the aim of making the software as user-friendly as possible. In this way, technical difficulties are minimized, thereby extending, so to speak, the "human compatibility" of the system. The type of information contained, the language used, and the structure of the Wiki are strongly oriented towards practical usage. The goal is not to illustrate reCluster's characteristics from a design or theoretical perspective, but solely to provide all the practical information and procedures necessary to make it functional and operational.

This document, therefore, does not offer any explanation or insight into what lies within reCluster, but rather provides only the technical knowledge needed to operate the system effectively. It is presented in a format that strives to be as accessible as possible to any user, along with the use of simple language and an attempt to minimize the amount of prior specific knowledge required. The intent of operating in this direction is to avoid discouragement due to the system's complexity for those who are approaching reCluster, allowing them to have adequate control over all usage phases right from the start.

The features of TiddlyWiki make it easy to distribute, publish, and use in an analytical manner. It proves to be an ideal tool for creating a practical knowledge source that is both accessible and easily updatable. In this way, it is possible to improve the documentation based on user feedback and to document future features that will be implemented within reCluster, perfectly aligning with the evolving and constantly updated nature of the project.

4 Conclusions

4.1 Limitations

In this section, I will analyze the limitations that the "reCluster" product is still subject to at the end of my thesis work, comparing them with previous ones. There are many areas where this project can expand, given its still quite early stage of development. Once reCluster enters a phase of practical use, it will undoubtedly become much easier to identify real problems and limitations. It will be possible to collect feedback from users, thus providing a complete picture of the actual issues encountered. Based on this data, shortcomings, possible improvements, and malfunctions can be identified. However, it is already possible to hypothesize some limitations that have emerged from my reflections upon completing my work. This section is divided into five subsections. In section 4.1.1, I discuss the impact of the lack of support for mobile devices. A simple calculation is provided to highlight a significant current discrepancy between the energy used in the lifecycle of these products and the energy required for their creation, which strongly contrasts with the sustainability values that reCluster aims to achieve. In section 4.1.2, a reflection is made on the issue of too many external software dependencies used in the implementation of reCluster and their negative consequences. In section 4.1.3, the total absence of the ability to modify any settings or parameters after installation is discussed, along with the flexibility and adaptability problems this limitation entails. In section 4.1.4, I suggest how the web dashboard I created represents a first step toward the realization of a centralized and integrated interface for cluster supervision. The absence of such a tool is also emphasized in terms of usability. Finally, in section 4.1.5, I explain the lack of technical documentation, which has been an obstacle in undertaking the path of modification and implementation of the solutions proposed in this thesis work.

4.1.1 Support for Mobile Devices

The first thought goes to the initial consideration regarding the goal of supporting the widest possible range of device classes. This feature has been greatly expanded, including new classes of devices that were previously unthinkable to use as useful hardware within a reCluster instance. However, many types of devices still cannot be used, with mobile devices, both Android and iOS, being the most prominent. Several considerations are necessary. First of all, the number of smartphones in circulation is three times that of desktop computers and laptops. [18] This opens the door to environments and use cases for reCluster different from the primary one it was designed for, such as universities. In fact, this type of device is more commonly found as a private object rather than as part of an academic institution's inventory. The situation is different for companies, which are equipped with corporate phones that are periodically replaced. This specific aspect

serves as an additional incentive to remove this limitation. The environmental impact of smartphones is increasingly evident and documented. Therefore, there is a need to shorten the path to the reuse of smartphones, trying as much as possible to extend their life cycle. [19]. The energy consumption of a smartphone throughout its lifecycle is significantly lower than its embodied energy, highlighting the importance of reuse strategies for such devices. On average, the embodied energy of a smartphone—covering raw material extraction, manufacturing, assembly, and shipping—ranges between 100 and 150 kWh [2, 20]. This accounts for about 85% of the total energy used over the device's entire lifecycle. In comparison, the operational energy consumption during a smartphone's typical three-year lifespan is much smaller, estimated at around 35 kWh in total, assuming a consumption of approximately 11.8 kWh per year [21]. The energy consumption for charging is derived from the average battery capacity, which is about 3,000 mAh for modern smartphones, multiplied by the voltage (3.8V) to yield 11.4 Wh per charge. With an average battery lasting around 500 full charge cycles during the device's lifespan [22], this gives an approximate total energy consumption for charging of 5.7 kWh. Even with the energy spent on networking, which adds about 10-15 kWh during the device's use [23], the energy used in the operational phase remains significantly lower than the energy spent during production. This emphasizes the relevance of integrating smartphones into systems like reCluster, as extending their life cycle can mitigate environmental impact and improve energy efficiency. An extension in this direction of reCluster would certainly help provide an additional opportunity for users to reintegrate their unused devices into a virtuous flow. This will help match the amount of energy used over the life cycle with the embodied energy of the device. It is also important to consider that, in recent years, the computational power of smartphones has increased significantly, while at the same time, a great deal of attention has been paid to efficiency [24]. These factors make mobile devices suitable and consistent with the goals of reCluster. On the other hand, one must imagine technical difficulties such as the impact continuous power supply may have on device batteries, which today are not easily removable.

4.1.2 Software Dependencies

Although reCluster has been built with the aim of limiting dependencies on external software, there is still significant room for improvement in this regard. The fact that a considerable number of external components are required limits both reliability and integrity, especially in the long term. Fewer dependencies mean more control over software operation and its sustainability over time, which also benefits security. A concrete example is the use of npm in the server environment. However, it is clear that beyond a certain point, reducing dependencies further becomes inefficient.

4.1.3 Inability to Modify Settings After Installation

Another evident problem that emerged during the development and testing work was the lack of the ability to modify any settings. There are no tools, commands, or interfaces that allow actions or modifications to the cluster structure or the settings of individual nodes. This deficiency makes reCluster particularly rigid, resulting in a blow to overall

usability. It is extremely inefficient, in fact, not to be able to change any settings after installation. This problem becomes more significant as the number of nodes within a cluster increases. Basic operations such as removing from the server, modifying the control interface, changing network settings, or updating the system must be guaranteed. The absence of these mechanisms is, however, understandable given the early stage of development in which reCluster currently lies.

4.1.4 Lack of an Integrated Control Interface

Another shortcoming closely related to the previous one is the lack of an easily accessible interface from which to monitor and operate the entire system. The web dashboard I implemented represents a first step in that direction, but it is limited to providing only a visualization of information without any ability to act on settings or perform any actions. Usability and practicality would greatly improve with the presence of an integrated and centralized interface from which to operate the system in all its capabilities. With a comprehensive and unified control panel, users would be greatly facilitated in their tasks, which would otherwise be complex and cumbersome.

4.1.5 Limitations in Working on reCluster

During the time before the implementation work and modification of the various software components, I greatly suffered from the lack of detailed technical documentation that clearly illustrated the role of each file, each function, and the logical execution flows. This applied to both the installation script and the server component. I had to spend a great deal of time reconstructing the software structure in detail, which was certainly a discouraging factor in acquiring the necessary knowledge to begin making modifications. I can, therefore, say that there is a definite lack of documentation aimed at developers working on the reCluster project, which is essential for minimizing the time required to become sufficiently familiar with its implementation structure and thus contribute modifications and improvements.

4.2 Future Work

In this section, I will discuss how future work on the reCluster project might be directed and which specific intervention areas it should focus on. I will propose ideas and deductions derived both from the analysis of the current software limitations and from insights and reflections that emerged during the implementation work. These will cover one or more areas or components of reCluster. This section is structured into five subsections. The topics are presented in ascending order according to an estimate of the complexity and effort required for implementation. In subsection 4.2.1, the need to implement a version of reCluster suitable for installation on mobile devices is briefly mentioned. Then, in subsection 4.2.2, I suggest creating a tool capable of designing and preconfiguring all system components to speed up and simplify the installation process. Subsection 4.2.3 proposes the implementation of an energy monitoring service to provide useful data and statistics for energy management in a cluster. Possible improvements for the button press device and its new developments are hypothesized in subsection 4.2.4. Finally, in

subsection 4.2.5, it is highlighted how the implementation of a web-based control interface, from which it is possible to monitor and govern every single aspect of the system, would be vitally important, with enormous positive consequences on usability.

4.2.1 Support for Mobile Devices

As already mentioned, one of the major current limitations of reCluster is the inability to employ mobile devices. Therefore, it would be appropriate and urgent to produce all the necessary software components to enable installation and use on both iOS and Android devices. The technical difficulties would be numerous, starting with the operating system. There would also be problems related to power-on strategies. However, given the great potential represented by these device classes, I believe that significant efforts should be made to support them, starting with the software ad the procedures that would make possible its installation on this clas of devices.

4.2.2 Integrated Configuration Tool

Currently, one of the most significant issues facing reCluster users is the large number of procedures with which they must become familiar to obtain a ready-to-use instance. Users must navigate between preparing configuration files, executing certificate generation scripts, generating configuration files, the installation process, resource-level cluster design, node configuration, and network settings. I imagine that having a dedicated tool to design a cluster and prepare everything automatically would be a great help. This tool would be as high-level as possible, equipped with an extremely intuitive graphical interface. Users could specify the nodes they want to add, their type, the power-on strategy, network settings, and so on. The software would produce custom disk images for each node, as well as firmware for any BPDs. All configuration files could be automatically generated based on the user's design specifications and pre-installed in the operating systems destined for the various nodes. The user would only need to install the correct software on the corresponding machine, as well as the firmware for the devices. In this way, the level of automation would be raised to a very high level. This would result in significant time and energy savings for reCluster operators.

4.2.3 Energy Monitoring Service

Given reCluster's inherent focus on energy efficiency, I believe it could be useful and meaningful to implement a service within the server that monitors the consumption of individual nodes along with the overall cluster. This would provide statistics and useful information for calculating values such as the overall energy cost or the system's efficiency level. This way, it is possible to instantly detect anomalies concerning consumption, as well as collect long-term data that allows for a clear picture of the situation. The system operator could then make specific decisions regarding the energy strategy and efficiency, based on the usage data of the nodes. Implementing this service within a web interface would allow for easy integration of highly effective tools, such as interactive charts.

4.2.4 Improvements and Developments for BPD

The button press device (BPD) offers great potential for development, enhancing the functionalities and capabilities of the first version. One initial improvement could consist

of designing a version of the device that allows for pressing side power buttons, present on some laptop models, as well as desktop computers. The architecture would remain unchanged from both a hardware and software perspective, but modifications would be made to the chassis design. So far, mobile devices have not been considered in the development of reCluster, but it is conceivable that a possible strategy for turning them on could be the use of a dedicated BPD. It is not inconceivable to create a single device capable of hosting multiple mobile devices and turning them on individually. Therefore, producing BPDs with different design variations will surely prove useful and beneficial.

Another modification that would bring practicality to the various BPDs' design would be the creation of a custom-designed control board. The use of dedicated electronic components, combined with the design of a board specifically for this purpose, would undoubtedly help reduce size and cost. More compact and economical devices could thus be obtained, which would certainly have fewer design and size constraints.

One possible significant improvement in hardware would be adding a battery, allowing the BPD to operate without power sources other than the node it needs to activate. If the node does not provide power via the USB port when turned off, the battery would allow the device to remain on standby, ready for use for a certain period, and to power on when required. Once the node is turned on, the battery could be charged via a USB port on the node. Further research and considerations would be necessary to assess the sustainability, convenience, and technical validity of this solution compared to providing an independent 5-volt power line dedicated to the BPD. It is important to remember that a battery, whatever its characteristics, will one day become electronic waste. Comparing, in these specific usage conditions, the lifespan of a battery versus that of a USB power supply would likely result in the latter being more sustainable. The same conclusion would be reached when considering the cost of the two items, as it is much easier to find a used but perfectly functioning USB power supply than a battery. Consequently, this choice would prove to be much more in line with reCluster's goal of reducing the impact of electronic waste.

The most sensible solution might be to provide support for using a battery while encouraging the use of a fixed power supply.

4.2.5 Web-Based Control Interface

The dashboard I implemented represents a good starting point, as it is a basic, immediate, and effective monitoring solution. However, as highlighted by the analysis of the limitations, it is still impossible to make any changes to the reCluster settings, either through a web interface or other methods. Therefore, I believe it will be very important to build a future interface that integrates all monitoring and management functionalities into a single place. It should be offered by the server, which is in a central position, allowing access to all necessary information. The presence of such a tool would greatly facilitate users' operations, significantly increasing the usability and ease of managing a reCluster cluster. The practicality resulting from the introduction of a complete and centralized control panel would certainly encourage the spread of reCluster in areas and contexts beyond academia, enabling the project to find applications where efficiency in use is a fundamental requirement.

The solutions I have proposed are just a few of the growth and expansion possibilities that a project like reCluster presents. I believe it would be essential to share all ideas and proposals among those working on the project to structure, prioritize, and design them coherently and non-fragmentarily.

4.3 Considerations

Following the conclusion of the proposed development work within this thesis, I am able to draw some conclusions regarding the current state of the reCluster project, as well as compare it with its past and future potential.

I believe that the reCluster project holds significant potential for development, and even in its early stages, it demonstrates much of its capabilities. Its political relevance, combined with sound technical implementation, makes this type of architecture suitable for a wide range of real-world applications. By continuing in the direction laid out by this work, it will be possible to significantly increase the effectiveness and overall quality of all the components that make up the product. It is important to consider that the number of people who have contributed to the development and creation of reCluster so far is very limited. The prospect of creating a real community and a dedicated team to further evolve the project is of considerable importance, especially given the uniqueness and the concrete values that the initial idea offers.

The creation of a tool like this is particularly relevant in light of the growing awareness within the IT sector regarding sustainability issues. Naturally, the wider the adoption and use of reCluster, the greater its impact in mitigating the negative effects of resource waste, both in terms of hardware and energy.

As repeatedly emphasized in this thesis, for software to succeed, it must meet certain characteristics. It would be more accurate to speak of an ecosystem rather than just software. As I have demonstrated, there are multiple possible directions for expansion, and software development is just one aspect. It is hoped that in the future, it will evolve into a complete set of solutions, programs, and devices capable of working in harmony to achieve the declared goals. This is an iterative process, and at the start, it is essential to establish characteristics that precisely define the type of use for which reCluster is intended.

This leads to a simpler and more accurate requirements engineering process, which is still unclear at this stage. A concrete example is the use of GraphQL for managing data access. This solution requires familiarity with a system whose architecture can initially seem difficult and complex. Using such a system may make sense if you imagine having thousands of nodes in a single cluster, where data access optimization is relevant. However, by analyzing the actual context, it becomes clear that the real maximum size of a reCluster instance may reach a hundred nodes at most. Thus, a more traditional solution, perhaps involving SQLite and simple REST APIs, not only offers simpler implementation and use but also has the advantage of reducing dependencies and resource consumption (such as memory) during execution. This underscores the importance of clearly and precisely defining the product's practical use case.

Another example, already mentioned in section 4.1.4, is the total absence of a tool

that allows for post-installation configuration changes. While such a system could be used, it is difficult to imagine it in a real production context. A simple example is the case where a node breaks down irreparably and needs to be removed, or where a power-on device breaks, and there is an urgent need to use the node it was associated with. The lack of any configuration and tuning mechanism risks making a reCluster instance practically unusable in a very short time. In general, it is essential that reCluster acquires flexibility to make it the right tool for a wide range of real-world cases.

To identify the best and most efficient interventions needed for the project, it is crucial that its use becomes more widespread, accompanied by efforts from developers to gather valuable feedback from users.

I can say that I am proud to have contributed to the improvement of reCluster. The work has been stimulating both from a research and deductive perspective—understanding what could be improved—and from a creative standpoint in designing the most suitable solutions to the identified problems. I believe that this project holds great political and practical value, and I hope it will achieve widespread success and realization of its objectives in the future, goals we should all believe in and hope for. I would be happy to contribute further to its development in the future.

Bibliography

[1] Angeli L. et al. Conceptualising resources-aware higher education digital infrastructure through self-hosting: a multi-disciplinary view. *LIMITS*, 2022.

[2] Lotfi Belkhir and Ahmed Elmeligi. Assessing ict global emissions footprint: Trends to 2040 & recommendations. *Journal of Cleaner Production*, 2018.

[3] K. et al. Parajuly. Future e-waste scenarios. *United Nations University*, 2019.

[4] Various Authors. Electronic ecologies conference. In *Electronic Ecologies Conference Proceedings*, 2019.

[5] BOINC Project. Boinc: Berkeley open infrastructure for network computing. *BOINC Project*, 2024.

[6] FUSS Project. Free upgrade for south tyrol's schools. *FUSS Project*, 2024.

[7] Biomodd Project. Biomodd: An art-science project integrating e-waste. *Biomodd*, 2024.

[8] Datacenterlight.ch. Decentralizing datacenters for green efficiency. *Datacenterlight*, 2024.

[9] Informatici Senza Frontiere. Bridging the digital divide through tech reuse. *Informatici Senza Frontiere*, n.d.

[10] Various Authors. Computing beyond limits conference. In *Computing Beyond Limits Proceedings*, 2021.

[11] Mark Mills. The wild wild waste. *ACM*, 2003.

[12] Vrious Authors. Global e-waste surging: Up 21 *UNU Press Release*, 2020.

[13] R. et al. Sharma. E-inventory for proactive e-waste management. *ACM*, 2010.

[14] S. et al. Ricciardi. Evaluating energy savings in wol-enabled networks of pcs. *Computing Systems*, 2017.

[15] R. et al. Narayan. The internet of tomorrow must sleep more and grow old. *IEEE*, 2016.

[16] J. et al. Baker. Technology replacement and avoiding obsolescence: Is it possible? *IEEE*, 1998.

[17] Vishwajeet Yadav, Savita Malik, Vikas Sharma, and Shaveta Madan. Significance of documentation in software development. *2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, 2014.

[18] Samsung Insights. Your phone is now more powerful than your PC. 2021.

[19] Anders S. G. Andrae and Tomas Edler Andersen. Life cycle assessments of consumer electronics: Smartphone, Laptop, Tablet. *The International Journal of Life Cycle Assessment*, 2015.

[20] Dom Galeon. Carbon footprint of your smartphone. *Futurism*, 2017.

[21] Christopher Calwell and Noah Horowitz. *Power Consumption in Smartphones: An Analysis.* NRDC, 2012.

[22] Battery University. Battery lifespan of smartphones, 2024.

[23] Gunther Auer, Valerio Giannini, et al. Energy efficiency in mobile networks: State of the art and perspectives. *Bell Labs Technical Journal*, 2011.

[24] Muhammad Furqan, Ammar Shehzad, et al. Mobile edge computing: A review of architectures, enabling technologies, and applications. *The Journal of Supercomputing*, 2020.

Appendix A Implementation Details

I will now illustrate some implementation details of the various interventions I carried out on reCluster components during my thesis work. This will primarily involve analyzing key pieces of code that are particularly significant from a design perspective and that I consider especially relevant for a thorough understanding of the technical details of the various modifications made. The section is structured following the content order of Chapter 3. The chapter begins with the explanation of *Sysbench support* in A.1, where I discuss the changes in the APKBUILD file and the handling of instruction sets. Then, in A.2, I describe the processes for managing network interfaces, which were key to ensuring reliable network configurations in reCluster. In A.3, I detail the various power-on strategies, including Wake-On-LAN and Smart Plug integration, along with the implementation of NodeService. Subsequently, in A.4, I delve into the Button Press Device (BPD) firmware and its role in automating node power-on. In A.5, I present the automatic removal of old installations to prevent conflicts. In A.6, I explain the automatic SSH certificate installation process. A.7 focuses on the specification of basic parameters via a graphical interface for improved user interaction, while A.8 concludes with the implementation of the web-based dashboard for monitoring and controlling reCluster nodes.

A.1 Sysbench Support

A.1.1 Update of the APKBUILD File in the Alpine Linux Repository

The following shows the modification made to the APKBUILD file in the Alpine Linux Community repository, regarding the addition of the `--without-gcc-arch` flag. The APKBUILD file contains the build directives and exists for each package included in the repository. In Alpine Linux release 3.19, which I used when I began development, the Sysbench package was compiled without this flag, leading to the backward compatibility issues discussed in the thesis.

A.1.2 CPUID Level and Instruction Set

The CPUID level under which compatibility problems occurred was level 11. In contrast, the i3-7100 processor, which did not have the issues, had a CPUID level of 22.

The two instruction sets that caused the "Illegal instruction" exception were:

- **AWX**: Introduced around 2011 (in fact, the problematic computer dates back to 2009/2010), it is a feature that provides significant improvements to workloads that can be vectorized, performing simultaneous operations on data in larger registers.
- **F16C**: Allows working with half-precision versions of floating-point numbers, thus saving memory at the cost of some precision. It was introduced in 2013 with the Haswell architecture.

A.1.3 Sysbench Source Code Analysis

Conducting an investigation at the Sysbench source code level, I focused on the auto-configuration part, involving autoreconf. I then analyzed the configure.ac file, which contains directives aimed at obtaining the best possible configuration for the current compilation environment.

I discovered that it contains the necessary directives to determine the most optimal compiler flags unless they are explicitly specified by the user through CFLAGS or the aforementioned `--without-gcc-arch` option is not used.

```
# Try to guess the most optimal compiler architecture flag, unless it's
    already
# been specified by the user via CFLAGS (or --without-gcc-arch is passed
    to
# configure)
AS_CASE([$CFLAGS],
  [*-march=*],,
  [AX_GCC_ARCHFLAG([no])]
)
```

Listing A.1: configure.ac architecture optimization flag detection.

In the case where the compiler is `gcc`, optimization flags are specified:

```
# First check for gcc and g++
if test "$GCC" = "yes"
then
  CFLAGS="-ggdb3 ${CFLAGS}"
  DEBUG_CFLAGS="-O0"
  OPTIMIZE_CFLAGS="-O3 -funroll-loops"
  GCOV_CFLAGS="-O0 --coverage"
  GCOV_LDFLAGS="-coverage"
  ASAN_CFLAGS="-fsanitize=address"
  ASAN_LDFLAGS="${ASAN_CFLAGS}"
  MSAN_CFLAGS="-fsanitize=memory"
  MSAN_LDFLAGS="${MSAN_LDFLAGS}"
fi
```

Listing A.2: configure.ac gcc compiler flags.

```
if test "$ax_cv_c_compiler_vendor" = "sun"
then
  isainfo_k=`isainfo -k`
  if test "$target_cpu" = "sparc"
  then
    MEMALIGN_FLAGS="-xmemalign=8s"
    IS_64="-m64"
    LDFLAGS="${LDFLAGS} -L/usr/lib/${isainfo_k} -L/usr/local/lib/${
    isainfo_k}"
  else
    if test "$isainfo_k" = "amd64"
    then
      IS_64="-m64"
      LDFLAGS="${LDFLAGS} -L/usr/lib/${isainfo_k} -L/usr/local/lib/${
    isainfo_k}"
    fi
  fi
```

Listing A.3: configure.ac architecture detection management.

However, these flags cover aspects such as debug information, compilation performance, code coverage (to precisely know which parts of the code are executed at a given time), and memory error management. Therefore, they have no implications related to the use of specific instruction sets concerning the type of processor in use. The configure.ac file also contains a section related to the management of specific architecture.

Specific flags are set based on the type of architecture detected. The distinction is made between SPARC and AMD64 architectures. Even in this case, influences related to the use of specific instruction sets within the AMD64 architecture emerged.

In the case of the first error, the method that analyzes the `/proc/cpuinfo` file is executed to find evidence of the source of the issue. In any case, the user is then prompted to execute the function responsible for compilation. On the other hand, if Sysbench is not installed, the user is prompted to attempt installation via a packet manager. If this fails or the user declines, the compilation is still offered, as well as in cases of other generic errors.

A.1.4 Analysis of the Sysbench Control Function

Within the installer, I added a specific function responsible for checking the presence and operational status of Sysbench, called `check_sysbench`. The core of its functionality is encapsulated within an `if-then-else` construct that handles the various Sysbench error codes, which are always intercepted within the function after running tests. Specifically, the function manages exit code 132, corresponding to the attempt to execute illegal instructions, and exit code 127, which is associated with the absence of the Sysbench executable on the system.

```
    if [ $exit_code -eq 132 ]; then
    whiptail --title "Error" --msgbox "Sysbench encountered an 'Illegal instruction' error. Looking for possible causes..." 10 78
    # Perform CPU's specs checking
    check_cpu_specs
    compile_sysbench
    # Command not found case (code 126)
  elif [ $exit_code -eq 127 ]; then
    whiptail --title "Error" --msgbox "Sysbench command not found. Please install it." 8 78
    # Ask the user if he wants to install Sysbench using the apk paceket manager, only if there is an internet connection.
    if wget --spider --quiet http://www.google.com; then
      if whiptail --title "Sysbench installation" --yesno "Do you want to install it using the apk packet manager?" 8 78; then
        # Install sysbench
        apk update
        apk add sysbench
        # Updating the sysbench binary path
        SYSBENCH_PATH="/usr/bin/sysbench"
        whiptail --title "Installation successful" --msgbox "Sysbench installation through apk packet manager succeded." 10 78
        # Call again the check_system method.
        check_sysbench
      else
        whiptail --title "Installation refused" --msgbox "The user has refused to install Sysbench through the apk packet manager. You can still proceed compiling Sysbench from source." 10 78
        compile_sysbench
      fi
    else
      # If there is no internet connection, notify the user to install sysbench offline.
      whiptail --title "Network Error" --msgbox "No internet connection, it is not possible to install Sysbench automatically using the apk packet manager. You can still proceed compiling Sysbench from source." 10 78
      compile_sysbench
    fi
  elif [ $exit_code -ne 0 ]; then
    whiptail --title "Error" --msgbox "Sysbench exited unexpectedly (exit code: $exit_code) because of currently unknown reasons. Please check the exit code. You can anyway try to proceed compiling Sysbench from source." 10 78
    compile_sysbench
  else
    whiptail --title "Info" --msgbox "Sysbench is present and working properly." 8 78
    INFO "Sysbench is present and working properly."
  fi

```

Listing A.4: `sysbench_check` function, if then else construct.

A.1.5 Analysis of the Sysbench Compile Function

The core operations performed within the `compile_sysbench` function, responsible for the compilation and possible installation of Sysbench, can be summarized in three instructions:

```
1         # Running the Sysbench autogen script
2         ./autogen.sh >/dev/null
3         # Configure
4         ./configure --without-mysql >/dev/null
5         # Make
6         make -j >/dev/null
7
```

Listing A.5: The three main instructions for Sysbench compilation.

These are the three steps prescribed by Sysbench's documentation. Note the specific inclusion of the `--without-mysql` option, which disables MySQL database support, a feature that is unnecessary for use in reCluster. Before executing the actual compilation operations, the necessary dependencies are attempted to be installed, and temporary directories are prepared. At the end of the compilation process, an interactive flow is implemented in which the user can choose whether or not to install the newly compiled Sysbench binary. The name of the executable file is "recluster_sysbench", so that can be distinguished from the pre-compiled one.

```
1         # Asking user if they want to install the working Sysbench into
      the system
2         if whiptail --title "Sysbench Installation" --yesno "Now a
      working instance of Sysbench is available. Do you want to proceed
      installing this on the system so that it will be available after the
      end of the installation? Otherwise it will be removed when the
      installation ends. (y/N)" 12 78; then
3           whiptail --title "Installation" --infobox "Installing the
      sysbench binary to the ~/bin/ location." 8 78
4           # Creating the installation folder inside the current user home
       directory
5           mkdir -p ~/bin/
6           # Moving the binary
7           cp ./src/recluster_sysbench ~/bin/
8           # Updating the sysbench binary path
9           SYSBENCH_PATH="$HOME/bin/recluster_sysbench"
10        else
11          whiptail --title "Temporary Installation" --infobox "The
      Sysbench binary will remain to the temporary location and will be
      removed after installation's ending." 8 78
12          # Updating the sysbench binary path
13          SYSBENCH_PATH="$_sysbench_tmp_directory/$_subdirectory_name/src
      /recluster_sysbench"
14        fi
15
```

Listing A.6: The interactive construct to ask the user about reClusterSysbench installation.

A.2 Network Interfaces Management

A.2.1 The Functions Interaction

A noteworthy aspect of the network interface management and configuration process within the reCluster installation script is how the functions are structured and cooperate, dividing tasks to achieve the final goal. There are four functions in question:

- **read_interfaces**: This function gathers the names and MAC addresses of the available network interfaces from the system. The collected data is then packaged into a JSON variable.
- **read_interfaces_info**: This function analyzes each interface among those present, extracting data regarding speed and Wake On Lan support status from the data collected by the `read_interfaces` function. It uses the `ethtool` command to achieve this. Iterating over all interfaces, and involving the user in the decision-making process, it produces a list of valid interfaces. Interfaces with invalid speed values are excluded. If an interface has the 'd' value for the Wake On Lan support flag, it offers the user the option to enable it by changing it to 'g'. For each interface, based on its condition and the type of node on which the installation is being performed, the user is informed and asked whether or not to include the interface in the list of valid ones. This operation is performed using the dedicated `update_valid_interfaces` function. In the end, the user is also asked to choose the control interface by calling the dedicated `select_interface` function. If no interface is selected as valid by the user, the system notifies the impossibility of proceeding and allows the selection process to restart.
- **select_interface**: Through a radio button selection panel, the user is asked to choose an interface to use as the control interface by the server. If only one interface is present, it will be preselected. This function modifies the JSON variable containing all valid interfaces, setting the "controller" variable of the selected interface to `true`.
- **update_valid_interfaces**: This simple function is responsible for adding a user-approved valid interface, along with all its attributes, to the JSON variable.

The diagram shows how the different functions interact and what roles they play.

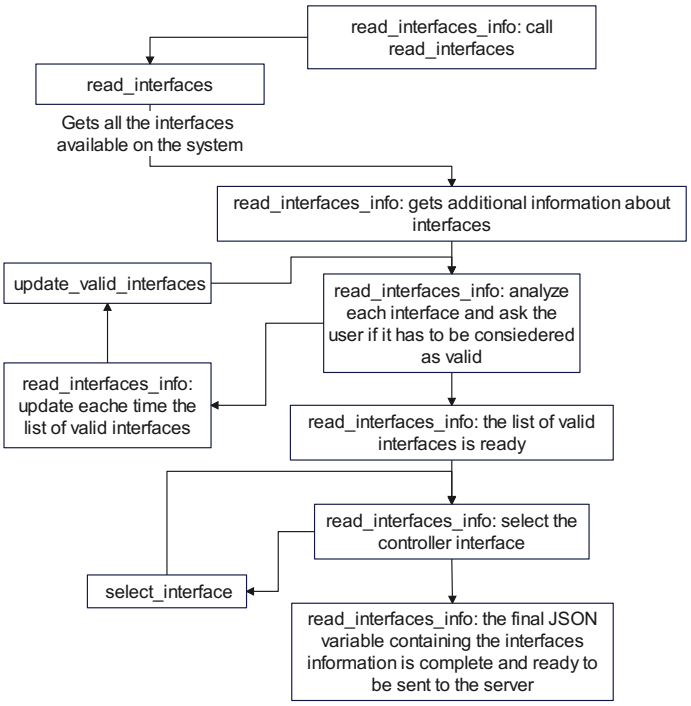


Figure A.1: The reCluster setup interaction between the interface management process' functions.

A.2.2 Some Excerpts From the Functions

Below are the main steps of the functions discussed in the previous subsection.

```
            read_interfaces() {
                ip -details -json link show |
                  jq \
                    '
                        map(if .linkinfo.info_kind // .link_type == "
    loopback" then empty else . end)
                      | map(.name = .ifname)
                      | map({address, name})
                    '
              }
```

Listing A.7: The `read_interfaces_info` function where available interfaces are collected through the `ip` command and then packed in a JSON variable using the `jq` command.

```
            ### Cycle over interfaces to obtain additional information ###
            echo "$_interfaces_info" | jq -r '.[] | "\(.name) \(.address)"' >$TMP_DIR/interfaces_info.txt
            while read -r _iname _address; do
              INFO "Processing $_iname with address $_address" # Name
              # Speed
              _speed=$($SUDO ethtool "$_iname" | grep Speed | sed -e 's/Speed://g' -e 's/[[:space:]]*//g' -e 's/b.*//')
              if [ -z "$_speed" ] || [ "$_speed" = "Unknown!" ]; then
                _speed=0
              elif [[ $_speed =~ [0-9]+[MG] ]]; then
                # If _speed is in the format '1000M' or '1G', extract the number
                _speed=$(echo $_speed | sed -e 's/[MG]//')
                DEBUG "Valid speed value $_speed read for interface $_iname ."
              else
                INFO "$_speed"
                whiptail --title "Speed value not valid." --msgbox "The speed value '$_speed' for the interface '$_iname' is not valid. The interface will be excluded." 10 60
                WARN "The speed value '$_speed' for the interface '$_iname' is not valid. The interface will be excluded."
                _speed=0
              fi
              _wol=$($SUDO ethtool "$_iname" | grep 'Wake-on' | grep -v 'Supports Wake-on' | sed -e 's/Wake-on://g' -e 's/[[:space:]]*//g')
              INFO "Wol: $_wol"

```

Listing A.8: The core loop inside the `read_interfaces_info` function where speed and WOL support info are read.

```
              # Build the whiptail command for the radio list
              whiptail_command="whiptail --radiolist \"Choose a controller network interface\" 20 100 10 $whiptail_args 3>&1 1>&2 2>&3"
              selected_interface=$(sh -c "$whiptail_command")
              # Verify if the user made a choice
              if [ $? -eq 0 ]; then
                INFO "You chose interface $selected_interface."
                # Update the "controller" field to "true" for the selected interface into the JSON array
                _valid_interfaces=$(echo "$_valid_interfaces" | jq --arg sel "$selected_interface" 'map(if .name == $sel then .controller = true else . end)')
                DEBUG "Updated valid interfaces' list: $_valid_interfaces"
              else
                if (whiptail --title "No interface chosen" --yesno "No interface was chose, and one is required to proceed with reCluster installation. Do you want to start over?" 10 60); then
```

```
12                    select_interface
13                  else
14                    FATAL "You can't proceed with reCluster installation
      without choosing an interface. Aborting..."
15                  fi
16                fi
17
```

Listing A.9: The controller interface selection process inside the `select_interface` function made using the Whiptail radio button dialog.

A.3 Additional Power-On Strategies

A.3.1 NodeService

I will focus on a fundamental component in the implementation of the new power-on strategies within reCluster, namely the "NodeService," which is part of the server. As previously mentioned, it also contains the two methods `boot` and `shutdown`, responsible for powering on and off the nodes, respectively. Below, the implementations of these two functions are illustrated based on the selected strategy.

Function `boot`

- In the case of the WOL strategy, the control interface is selected, a check is performed regarding the Wake On Lan support, and the appropriate function is invoked to send the packet, located in the dedicated service. The MAC address of the network interface is passed as an argument.

```
1            case 'WOL':
2            const controllerInterface = node.interfaces.find(intf =>
     intf.controller);
3            if (!controllerInterface) {
4              throw new NodeError(`Node '${args.where.id}' does not
    have a controller interface with WoL capability`);
5            }
6            // Wake on LAN strategy using the controller interface
7            await this.wolService.wake({
8              mac: controllerInterface.address,
9              opts: {
10               ...(process.platform === 'win32' && { address: node.
    address })
11             }
12           });
13           break;
14
```

- In the case of the AO strategy, no action is performed, as it is assumed that the node is already powered on.

```
1            case 'AO':
2            // Always On strategy
3            logger.info(`Node '${args.where.id}' power on strategy
    is configured as Always On (AO). Nothing to do.`);
4            break;
5
```

- In the case of the SP and BPD strategies, the presence of a power-on device associated with the node is checked, and then the methods for performing the power-on are invoked in their respective services.

```
            case 'SP':
            // Smart Plug strategy
            if (!node.powerOnDevice || node.powerOnDevice.deviceType
 !== 'SMART_PLUG')
              throw new NodeError(`Node '${args.where.id}' does not
have a valid Smart Plug configured.`);

            // Add logic to handle Smart Plug power on
            await this.smartPlugService.powerOn(node.powerOnDevice.
address, args.where.id);
            break;

          case 'BPD':
            // Button Press Device strategy
            if (!node.powerOnDevice || node.powerOnDevice.deviceType
 !== 'BUTTON_PRESS')
              throw new NodeError(`Node '${args.where.id}' does not
have a valid Button Press Device configured.`);

            // Add logic to handle Button Press Device power on
            await this.buttonPressDeviceService.pressButton(node.
powerOnDevice.address, args.where.id);
            break;

```

Function `shutdown`

- In the case of the AO strategy, no action is performed since the node cannot be turned off.

```
              case 'AO':
              // Always On strategy
              logger.info(`Node '${args.where.id}' is configured
as Always On (AO). It will not be shut down.`);
              break;

```

- In the case of the SP strategy, the presence of a smart plug associated with the node is checked in the data structure. Then, the `sudo poweroff` shutdown command is executed via SSH. At this point, the auxiliary method `waitForNodeToBeOff` is invoked, which waits for the node to stop responding to the ping requests sent by the server. Once the node is no longer reachable, a 15-second wait is applied to ensure the shutdown is complete. After this time, the method responsible for powering off the smart plug, located within its service, is called.

```
            case 'SP':
            // Smart Plug strategy
            if (!node.powerOnDevice || node.powerOnDevice.deviceType
     !== 'SMART_PLUG') {
              throw new NodeError(`Node '${args.where.id}' does not
    have a valid power on smart plug configured.`);
            }

            {
              // Perform shutdown via SSH
              const ssh = await SSH.connect({ host: node.address });
              await ssh.execCommand({
                command: 'sudo poweroff',
                disconnect: true
              });

              // Wait for node to be off
              await this.waitForNodeToBeOff(node.address);

              // Wait after ping unreachability for 15 seconds
    before turning off the smart plug to be sure that the node is
    really powered off gracefully
              await new Promise((resolve) => setTimeout(resolve,
    15000));
              await this.smartPlugService.powerOff(node.
    powerOnDevice.address, args.where.id);
            }
            break;

```

- In the case of the WOL and BPD strategies, the `sudo poweroff` shutdown command is simply executed via the SSH service.

```
        case 'WOL':
        case 'BPD':
            const ssh = await SSH.connect({ host: node.address });
            await ssh.execCommand({
              command: 'sudo poweroff',
              disconnect: true
            });

```

A.4 Button Press Device

A.4.1 Firmware

The firmware for the power button press device ideally handles two tasks: connecting to the network and executing actions corresponding to requests on the endpoints. Below, I will analyze the management of these two activities.

Network Connection

The procedure is distributed between the `setup` function and the `loop` function in the firmware. Within the `loop` function, instructions are simply reiterated cyclically to restore the connection in case of disconnection. To connect to the specified WiFi network during the configuration process, the `ESP8266WiFi` library is used. The connection is initialized with the `begin` method, which takes the SSID and password as arguments. A maximum of one minute is allowed for the connection before declaring failure. During this time, an informational message is printed to the serial port every second. A corresponding message is also printed in case of a failed connection. These messages are read by the configuration script for debugging purposes. Once the device is connected to the WiFi network, the network connection configuration proceeds. The gateway IP address and subnet mask are automatically retrieved. In case of a configuration failure, the corresponding message is printed to the serial port.

```
1     WiFi.begin(ssid, password);
2     unsigned long startAttemptTime = millis();
3     while (WiFi.status() != WL_CONNECTED && millis() - startAttemptTime <
      60000)
4     {
5       delay(1000);
6       Serial.println("Connecting to WiFi...");
7     }
8     if (WiFi.status() != WL_CONNECTED)
9     {
10      Serial.println("Failed to connect to WiFi");
11    }
12    else
13    {
14      if (!WiFi.config(local_IP, WiFi.gatewayIP(), WiFi.subnetMask()))
15      {
16        Serial.println("STA Failed to configure");
17      }
18      Serial.println("Connected to WiFi");
19    }
20
```

Listing A.10: The network connection procedure inside the BDP firmware.

Endpoint Management

As previously mentioned, there are three endpoints. Here is their detailed implementation:

- `/areyoualive`: Simply responds with HTTP status 200 OK.

```
                server.on("/areyoualive", HTTP_GET, [](
    AsyncWebServerRequest *request)
                { request->send(200, "text/plain", "OK");
                  Serial.println("Are you alive request received");
    });
```

- `/test`: Collects the parameters from the request (specified via URL) and validates them. Then, it performs the corresponding servo motor movement, waits for the specified time, and returns the servo to its initial position. It responds accordingly with the correct HTTP codes, i.e., 200 on success and 400 for malformed requests.

```
            server.on("/test", HTTP_GET, [](AsyncWebServerRequest *
    request)
            {
              if (request->hasParam("angle") && request->hasParam("
    duration")) {
                String angleParam = request->getParam("angle")->
    value();
                String durationParam = request->getParam("duration")
    ->value();
                int angle = angleParam.toInt();
                int duration = durationParam.toInt();
                if (angle >= 0 && angle <= 180) {
                  servo.write(angle); // Move the servo to the
    specified angle
                  Serial.printf("Moving servo to %d degrees\n",
    angle);
                  delay(duration); // Wait for the specified
    duration (in milliseconds)
                  servo.write(0); // Return to 0 degrees (initial
    position)
                  Serial.printf("Returning servo to 0 degrees after
    %d ms\n", duration);
                  request->send(200, "text/plain", "Servo moved
    successfully");
                } else {
                  request->send(400, "text/plain", "Invalid angle.
    Must be between 0 and 180.");
                }
              } else {
                request->send(400, "text/plain", "Missing angle or
    duration parameter.");
              }
            });
```

- `/press`: Performs the same actions as the `/test` endpoint, but based only on the fixed values defined in the appropriate variables, set during the device configuration and integrated into the firmware.

```
            server.on("/press", HTTP_GET, [](AsyncWebServerRequest *request)
            {
              // Move the servo to the specified angle and keep it for the duration
              servo.write(turningDegrees);  // Move to specified degrees
              Serial.printf("Servo moved to %d degrees\n", turningDegrees);
              delay(pressDuration); // Keep it in the position for the specified duration
              servo.write(0); // Move back to 0 degrees
              Serial.printf("Servo returned to 0 degrees after %d ms\n", pressDuration);
              request->send(200, "text/plain", "SUCCESS");
              Serial.println("Button press request received"); });
```

A.4.2 Configuration Script

Regarding the configuration script, I include only the main function, where the logical execution flow described in the corresponding chapter is visible.

```
main() {
    ERROR_THROWN=false

    echo "PARAMETERS COLLECTION - Collecting parameters..."
    collect_parameters

    echo "SERIAL PORT DETECTION - Detecting serial port..."
    port=$(arduino-cli board list | grep "tty" | awk '{print $1}')
    if [ -z "$port" ]; then
        whiptail --msgbox "No serial port found. Check the USB connection and make sure the serial port is not busy or invisible." 10 60 --title "Serial Port Error"
        handle_restart_option "Do you want to start over with the installation?"
    fi
    echo "SERIAL PORT DETECTION - Serial port detected: $port"
    install_libraries
    generate_code "$ssid" "$password" "$local_IP" "$servoPin" "$turningDegrees" "$pressTime"
    upload_code "$port"
    serial_output_file="serial_output.txt"
    echo "SERIAL PORT MONITORING - Monitoring board serial port output for 60 seconds...hold on"
    rm -rf "$serial_output_file"
    record_serial_output "$port" "$serial_output_file"
    if [ $? -ne 0 ]; then
```

```
22              handle_restart_option "The serial port was disconnected.
    Please check the connection and restart the installation."
23          fi
24          echo "SERIAL PORT MONITORING - Serial port monitoring completed."
25           check_serial_output "$serial_output_file"
26          result=$?
27          if [ "$result" -eq 1 ]; then
28              whiptail --msgbox "No error messages found in the serial
    output" 8 39 --title "Success"
29          elif [ "$result" -eq 2 ]; then
30              handle_restart_option "Device network error: STA Failed to
    configure. Check network settings. Do you want to restart the
    installation?"
31          elif [ "$result" -eq 3 ]; then
32              handle_restart_option "Device network error: failed to
    connect to WiFi. Check network settings. Do you want to restart the
    installation?"
33          elif [ "$result" -eq 4 ]; then
34              handle_restart_option "No output was read from the serial
    port. There may be an issue with the connection. Do you want to
    restart the installation?"
35          elif [ "$result" -eq 0 ]; then
36              echo "DEVICE TESTING - Testing device functionality over WiFi
    ..."
37
38              if check_device "$local_IP"; then
39                  repeat_motor_test "$local_IP"
40              else
41                  handle_restart_option "Device network error: Unable to
    contact the device over WiFi. Check network settings. Do you want to
    restart the installation?"
42              fi
43          else
44              handle_restart_option "It was not possible to get the board
    status. Do you want to restart the installation?"
45          fi
46      }
47
```

Listing A.11: The main function of the BPD configuration script's main function, where the are performed all the operations up to the serial port monitoring after firmware uploading.

The most significant part from a design perspective is the management of various exceptions along the way, with consequent notification and interaction management for the user.

A.4.3 Design and 3D Printing

Regarding the design, I provide a brief note about the software used for 3D modeling and slicing during the printing phase.

To create the CAD drawing, I used Fusion 360, a very powerful and widely used next-generation software developed by Autodesk. It is free for personal use and integrates a large number of powerful tools. Two standout features are the design timeline and the ability to perform in-app rendering.

For slicing and preparing for 3D printing, I used the very popular open-source software Cura. It allows the configuration of a vast array of settings and supports a large number of machines. It is constantly updated. After importing the 3D models from Fusion 360, I proceeded with generating the machine code, which was then executed by the printer during the printing phase.

The printer used was a Creality Ender 3 V3 KE. It is a next-generation consumer filament printer, offering particularly high print quality and speed.

Figure A.2: The Creality Ender 3 printer used to print the BPD frame.

A.5 Automatic Removal of Old Installations

The implementation mechanism of this automation is quite simple. A check is performed at the beginning of the installation script execution, regarding the presence of the reCluster installation folders. Everything is implemented within the `check_directories` function, which is located inside the main `verify_system` function and is responsible for verifying system conditions before proceeding with the reCluster installation.

If the folders are present, a choice dialog created with Whiptail is displayed, offering the user the option to run the uninstallation script. If the user agrees, the script is executed by calling the `recluster.uninstall.sh` command, which is available as a system-level command. Once the uninstallation is complete (automatically removing everything that needs to be removed depending on what is left from the previous installation), the installation script proceeds normally with its execution, failing if any old installations are detected.s

```
1  check_directories() {
2    if [ -d "$RECLUSTER_ETC_DIR" ] || [ -d "$RECLUSTER_OPT_DIR" ]; then
3      # Use whiptail to ask the user
4      if whiptail --title "Directories Exist" --yesno "Directories '
     $RECLUSTER_ETC_DIR' and/or '$RECLUSTER_OPT_DIR' already exist. Do you
     want to run 'recluster.uninstall.sh' to remove them and retry?" 10 80;
      then
5        # User agreed to uninstall, run the uninstall script
6        INFO "Running uninstall script 'recluster.uninstall.sh'..."
7        recluster.uninstall.sh || FATAL "Failed to run 'recluster.
     uninstall.sh'."
8        # Retry the directory check after uninstall
9        if [ -d "$RECLUSTER_ETC_DIR" ] || [ -d "$RECLUSTER_OPT_DIR" ];
     then
10         FATAL "Directories still exist after running uninstall script."
11       fi
12     else
13       # User chose not to uninstall, exit with a fatal error
14       FATAL "Directories '$RECLUSTER_ETC_DIR' and/or '
     $RECLUSTER_OPT_DIR' already exist, and the user opted not to remove
     them."
15     fi
16   fi
17 }
```

Listing A.12: The `check_directories` function.

A.6 Automatic Installation of the SSH Certificate

The functionality that allows the user to choose whether to automatically download and install the SSH certificate is encapsulated in the `automatic_install_ssh_certificate` function, which is the last one invoked during the execution of the reCluster installation script. First, it checks the type of node on which the installation is being performed. If the `INIT_CLUSTER` flag is set to "true," no procedure is executed, and only a message is printed, notifying that SSH certificate installation is not necessary on a controller node. The installation then concludes.

Otherwise, a choice dialog created with Whiptail is displayed, where the user can choose whether to proceed or not. A conditional construct is implemented to handle both the user's response and the various success and failure cases. All corresponding informational dialogs are present within this construct. Initially, the address from which to download the certificate is prepared by concatenating the server's IP address. The download is then executed using the `wget` command. The certificate's content is inserted into the `~/.ssh/authorized_keys` file of the node on which the installation is being performed. In case the download or installation fails, the corresponding dialogs are always displayed, reminding the user to manually check the installation status of the certificates.

```
1  automatic_install_ssh_certificate() {
2    # Check if it's a controller node
3    if [ "$INIT_CLUSTER" = true ]; then
4      INFO "Since it is a controller, no automatic certificate
   installation is required."
5      return
6    fi
7    # Modify the $_server_url to remove the port, the /graphql, and the
   http:// prefix if necessary
8    clean_server_url=$(echo "$_server_url" | sed -e 's|http://||' -e 's
   |:[0-9]*/graphql$||')
9    # Ask the user if they want to automatically download and install the
    certificate
10   if whiptail --title "SSH Certificate Installation" --yesno "Do you
   want to automatically download and install the SSH certificate from
   the server?" 10 60; then
11     # Attempt to download the certificate using wget
12     INFO "Downloading SSH certificate from 'http://$clean_server_url
   :3000/api/getsshcertificate'"
13     cert_content=$(wget -qO- "http://$clean_server_url:3000/api/
   getsshcertificate")
14     # Check if the download was successful
15     if [ $? -eq 0 ] && [ -n "$cert_content" ]; then
16       INFO "Installing server's downloaded SSH certificate to '
   $_ssh_authorized_keys_file'"
17       if ! printf "%s\n" "$cert_content" | $SUDO tee -a "
   $_ssh_authorized_keys_file" >/dev/null; then
18         WARN "Error appending SSH certificate. Please manually verify
   the correct certificate is installed."
19         whiptail --title "SSH Certificate Append Error" --msgbox "There
```

```
    was an error appending the SSH certificate. Default certificates from
    config files installed. Please manually verify that the correct
   certificate is installed." 10 60
20         fi
21       else
22         # Notify the user via whiptail dialog that the download failed
23         WARN "Failed to download SSH certificate from the server. Please
   manually verify the correct certificate is installed."
24         whiptail --title "SSH Certificate Download Failed" --msgbox "The
   automatic download of the SSH certificate from the server failed.
   Default certificates from config files installed. Please manually
   verify that the correct certificate is installed on this node." 10 60
25       fi
26     else
27       # User refused the automatic installation
28       whiptail --title "SSH Certificate Installation Skipped" --msgbox "
   You have refused to automatically download and install the SSH
   certificate. Default certificates from config files installed. Please
   verify that the correct certificate is installed on this node." 10 60
29       WARN "User skipped automatic SSH certificate installation. Default
   certificates from config files installed. Please manually verify the
   correct certificate is installed."
30     fi
31   }
```

Listing A.13: The `automatic_install_ssh_certificate` function, that envelopes all the automatic SSH certificate installation procedures.

A.7 Specification of Basic Parameters via Graphical Interface

The behavior of the script regarding argument parsing has been modified. All parameters, including those I identified as mandatory, are associated with global variables within the reCluster installation script. If an argument was not specified, the default value to which the variable was initialized would have been used. I removed the default value initialization for the set of basic parameters I chose, for which it is mandatory to specify a value.

I then introduced a check on the initialization of the variables after parsing the arguments. If these parameters were not specified, the variables would remain uninitialized. In that case, only for the parameters that were not specified, Whiptail dialogs appear, allowing the user to enter or select different values. For each entered value, validation is performed, unless it is a selection between predefined options. Both the validation functions and the controls responsible for displaying the input dialogs for parameters were placed inside the main `parse_args` function, which is the first function to be called during the execution of the installation script.

```
  if [ -z "$LOG_LEVEL" ]; then
  select_log_level
fi
if [ -z "$ADMIN_PASSWORD" ]; then
  ADMIN_PASSWORD=$(validate_non_empty "$ADMIN_PASSWORD" "Enter the Admin
    password:")
fi
if [ -z "$AUTOSCALER_PASSWORD" ]; then
  AUTOSCALER_PASSWORD=$(validate_non_empty "$AUTOSCALER_PASSWORD" "Enter
    the Autoscaler password:")
fi
if [ -z "$PC_DEVICE_API" ]; then
  while [ -z "$PC_DEVICE_API" ]; do
    ip=$(whiptail --inputbox "Enter the IP address for the PC Device API:
    " 8 78 --title "PC Device API IP" 3>&1 1>&2 2>&3)
    [ $? -eq 0 ] || exit 1
    if validate_ip "$ip"; then
      build_device_api_url "$ip"
    fi
  done
fi
if [ -z "$CONFIG_FILE" ]; then
  config_input=$(validate_non_empty "$CONFIG_FILE" "Enter the
    configuration file path:")
  while ! validate_path "$config_input"; do
    config_input=$(validate_non_empty "$CONFIG_FILE" "Enter a valid
    configuration file path:")
  done
  CONFIG_FILE="$config_input"
fi
if [ -z "$K3S_CONFIG_FILE" ]; then
  k3s_input=$(validate_non_empty "$K3S_CONFIG_FILE" "Enter the K3S
    configuration file path:")
  while ! validate_path "$k3s_input"; do
    k3s_input=$(validate_non_empty "$K3S_CONFIG_FILE" "Enter a valid K3S
    configuration file path:")
  done
  K3S_CONFIG_FILE="$k3s_input"
fi
if [ -z "$INIT_CLUSTER" ]; then
  if (whiptail --title "Initialize Cluster" --yesno "Do you want to
    initialize the cluster?" 8 78); then
    INIT_CLUSTER=true
  else
    INIT_CLUSTER=false
  fi
fi
```

Listing A.14: The controls into `parse_args` function, differentiated by the type of parameter.

A.8 Web Dashboard

The web interface has been implemented as previously mentioned with a client-side and a server-side. Let us take a closer look at how it has been built. We will analyze the DOM structure of the web page, some parts of the script responsible for the dynamic generation of the page, as well as the server-side components that serve the interface and retrieve data from the database.

A.8.1 The Web Page Structure

The DOM is quite minimal and linear. At the top, there is a block containing the title, as well as the colored indicator and the text showing the connection status with the server and the data update status. The rest of the page consists of a container block for the nodes. This container is populated with blocks containing individual nodes. Each block containing a node can have two or three sections depending on the presence of a power-on device, which is displayed in a dedicated section.

Each block contains a table with columns that are appropriate for representing the type of data. The header contains the CSS section that defines the style of the entire page. At the bottom of the body, there is also a section containing the JavaScript script responsible for generating the dynamic content. This script will be analyzed in the following section.

A.8.2 The Script To Generate Dynamic Content

The script is responsible for periodically fetching data about nodes from an API and updating the user interface to display relevant information. It makes use of asynchronous functions to ensure that the page remains responsive while fetching data from the server.

The function `fetchNodes()` retrieves the node data and dynamically creates HTML elements to represent each node in a structured way. This includes details such as the node's ID, name, status, IP address, and roles. The status is displayed with a color-coded circle: green for active states, red for unknown states, and orange for other statuses.

In addition to node details, the code also handles displaying information about network interfaces and power-on devices if they are available. The nodes are sorted to prioritize those with the `RECLUSTER_CONTROLLER` role.

Another function, `updateStatusIndicator()`, manages the status indicator displayed on the page, showing whether the data fetch process is working correctly or has failed. If more than 11 seconds have passed since the last successful fetch or if an error occurs, the indicator turns red, otherwise it remains green.

When the page loads, `fetchNodes()` is called immediately and then every 10 seconds to keep the data updated. The status indicator is also checked every second to ensure it reflects the current state of the data retrieval process.

```
    <html lang="en">
    <head>
    <style></style>
    </head>
    <body>
      <div class="container">
        <h1>reCluster Nodes Dashboard</h1>
        <h2>
          Refreshing status:
          <div
            class="status-indicator"
            id="statusIndicator"
            style="background-color: red"
          ></div>
          <span class="status-text" id="statusText">Failed</span>
        </h2>
        <div id="nodesContainer">
          <div class="node-block">
            <div class="title">Node</div>
            <table class="node-table">
            </table>
            <div class="title">Interfaces</div>
            <table class="interface-table">
            </table>
          </div>
          <div class="node-block">
            <div class="title">Node</div>
            <table class="node-table">
            </table>
            <div class="title">Interfaces</div>
            <table class="interface-table">
            </table>
            <div class="title">Power On Devices</div>
            <table class="poweron-table">
            </table>
          </div>
        </div>
      </div>
      <script>
      </script>
    </body>
  </html>

```

Listing A.15: The DOM structure of the reCluster web dashboard, with two nodes represented.

A.8.3 The Server Side Components

The server-side components are two:

- **Web server**: Written in NodeJS and part of the reCluster server, it is responsible for defining all the endpoints, as well as instantiating the actual web application through the Express plugin. It also handles the serving of the static HTML file containing the dashboard at the correct endpoint. The web server is then started in the main file of the application.

```
1         import express from 'express';
2         import path from 'path';
3         import nodesRouter from './routes/nodes';
4         import certificateRouter from './routes/certificate';
5         const webApp = express();
6         // API Routes
7         webApp.use('/api', nodesRouter);
8         webApp.use('/api', certificateRouter);
9         // Serve the status HTML page at the "/dashboard" route
10        webApp.use('/dashboard', express.static(path.join(__dirname,
      '../../../public')));
11        webApp.get('/dashboard', (req, res) => {
12            res.sendFile(path.join(__dirname, '../../../public', '
    dashboard.html'));
13        });
14        webApp.get('/', (req, res) => {
15          res.redirect('/dashboard');
16        });
17        export default webApp
18
```

- **Router**: It is responsible for executing the corresponding method in Prisma to retrieve all the necessary information about the nodes and provide it in JSON format when a request is made to the corresponding endpoint. This request is periodically executed by the JavaScript code on the page, which takes care of populating it with dynamic and updated content.

```
            import { Router } from 'express';
            import { prisma } from '~/db';
            const nodesRouter = Router();
            nodesRouter.get('/nodes', async (req, res) => {
              try {
                const nodes = await prisma.node.findMany({
                  include: {
                    status: true,
                    powerOnDevice: true,
                    interfaces: true // Include interfaces in the
    response
                  }
                });
                res.json({ nodes });
              } catch (error) {
                console.error('Error fetching nodes:', error);
                res.status(500).json({ error: 'Error fetching nodes'
     });
              }
            });
            export default nodesRouter;
```