University of Passau

Department of Informatics and Mathematics

Master's Thesis

Energy and Performance Evolution of Configurable Systems: Case Studies and Experiments

Author:

Niklas Werner

September 26, 2019

Examiners:

Prof. Dr. Sven Apel	Prof. Dr. Gordon Fraser
Chair of Software Engineering I	Chair of Software Engineering II

Advisor:

Christian Kaltenecker

Chair of Software Engineering I

Werner, Niklas:
Energy and Performance Evolution of Configurable Systems: Case Studies and Experiments
Master's Thesis, University of Passau, 2019.

Abstract

Contemporary software systems are often highly configurable and additionally they change over time (*evolve*). These changes do not only affect the functionality but also non-functional properties such as performance and energy consumption. While performance has always been important and in the focus of optimizations, recently energy consumption is becoming increasingly more relevant. However, reducing energy consumption while maintaining or even improving performance at the same time is not trivial and requires an understanding of the relation between performance and energy consumption and their behaviour with respect to software evolution. Existing studies in this field have either considered only the performance but not the energy consumption or have compared performance and energy consumption without considering the aspect of evolution. In this thesis, we combine these aspects in an exploratory manner by measuring and evaluating the performance and energy consumption of different releases and configurations of four case studies – *HSQLDB*, *Apache httpd*, *PostgreSQL* and *libvpx VP8*. For the evaluation, we directly compare performance and energy consumption and also investigate influences of specific configuration options on the performance and energy consumption. Additionally, we consider the correlation between performance and energy consumption. We find that there are changes in performance and energy consumption over the course of time and that changes equally affect performance and energy consumption. We are also able to attribute changes to specific configuration options in some cases. Additionally, we find that there are changes in the correlation, but we cannot determine a clear relation between changes in the correlation and specific configuration options.

Contents

List of Figures

List of Tables

1. Introduction

Nowadays, software systems are complex. Not only are they often highly configurable, offering a vast number of configuration options, allowing the software to be adapted to many different scenarios. They are also not static but are updated over time, they change. Software is said to *evolve*. These changes do not only affect the codebase and functionality of software, but they can also have an impact on non-functional properties, such as performance or energy consumption.

Performance in software systems has always been important, be it for high-performance computing or simply for user experience. Who wants to wait for a program to produce some result? With an ever increasing demand for computing power, there is also more and more energy consumption from computers. In contrast to that, the reduction of energy consumption as a whole is becoming more and more important. Reducing energy consumption while maintaining a high performance is not trivial for evolving configurable software systems. Every single configuration option and every single change can not only extend or modify the functionality of the software system, but can potentially also influence the non-functional properties. Such optimizations require an understanding of the relation between these two non-functional properties and their evolutionary behaviour.

There has been previous work in this field, investigating specific aspects of this topic. Siegmund et al. [SGAK15] investigate the performance in configurable software systems without considering the aspects of energy consumption or software evolution. In another related study, Tsirogiannis et al. [THS10] are concerned with the connection between performance and energy consumption without explicitly considering configurable systems or evolution. In our own previous work [Wer17], we investigated interactions of performance and energy consumption in configurable systems, combining the previous two studies while still not considering evolution. Finally, Hasreiter [Has19] combined the aspect of evolution with performance in configurable systems, but without considering the energy consumption.

This thesis builds upon our own previous work and the work of Hasreiter to combine all of these aspects: We investigate the evolution of performance and energy consumption in configurable systems. We measure the performance and energy consumption of four case studies, the database servers *HSQLDB* and *PostgreSQL*, the web server *Apache httpd* and the video encoder *libvpx VP8*. We then evaluate our measurement results to answer the question of whether and how performance and energy consumption of software systems, as well as the relation between those two non-function properties, change over time. For this evaluation, we investigate the measured performance and energy consumption directly, employ with *influence models* a modelling approach introduced by Siegmund et al. [SGAK15] to investigate the impact of specific configuration options on performance and energy consumption, and additionally consider the correlation between these two non-functional properties.

This thesis is structured as follows:

First, we present an overview on related work in Chapter 2. There we demonstrate how this work relates to other studies related to this subject, in particular the aforementioned ones.

Then, in Chapter 3, we provide relevant background information and introduce terminology used throughout this thesis, starting with different types of software, configurable software and software evolution. We define what we consider as performance and energy consumption, and describe influence models and the Pearson correlation, which we use as part of our operationalization.

Afterwards, we describe our methodology in Chapter 4. First, we introduce our research questions. Then, we describe our case studies and experimental setup. Finally, we explain the operationalization, detailing how we answer the research questions.

In the evaluation in Chapter 5, we answer the research questions one by one, presenting and discussing the results for each.

Having presented the results, we dedicate Chapter 6 to our consideration regarding both the internal and external validity of our findings.

Finally, we conclude this thesis in Chapter 7 with a summary, and outline ideas for possible future work.

2. Related Work

In this chapter, we provide an overview on work related to one or more of the aspects of this thesis. We point out differences between this thesis and the related work and at the same time motivate this thesis.

Performance and Energy Consumption

There have been several studies comparing performance and energy consumption.

Tsirogiannis et al. [THS10] analyse the energy efficiency of database systems. They measure the performance and energy consumption of so called 'micro-benchmarks', that they developed specifically for this purpose, and additionally specific algorithms that are used by the popular database system PostgreSQL. Energy consumption is measured for individual hardware components rather than the system as a whole, with the evaluation being focused on the energy consumption of the CPU. The comparison is performed between performance and energy efficiency, which are both defined to include the amount of work done. The measurements are performed with different combinations of software and hardware configurations. Their findings indicate a correlation between performance and energy efficiency.

In this work, we take a less fine-grained approach by measuring energy consumption of the whole system and using complete software systems, rather than only 'micro-benchmarks'.

Another study, by Costa et al. [CAKLR11], also compares performance and energy efficiency. The authors define a notion of performance and energy efficiency. Energy consumption is measured for the whole system, as opposed to measuring for individual components like in the study by Tsirogiannis et al. The study independently determines break-even points in data deduplication systems for performance and energy efficiency. Similar to the previously described study, different hardware configurations are used. They find that newer hardware systems are increasingly more power-proportional, i.e. the power consumption is proportional to the resource utilization. They conclude that tradeoffs exist between performance and energy optimizations for such newer hardware.

We perform a more direct comparison between performance and energy consumption. The hardware we use for our measurements falls into the category of newer hardware as described by Costa et al.

Finally, Xu et al. [XTW10] evaluate the performance and power consumption of database systems to find power–performance tradeoffs. They use PostgreSQL with a benchmark based on TPC. Power consumption and energy consumption are measured for the whole system and are compared directly with run time. They do not use a notion of performance or of efficiency. The comparisons are performed for multiple workloads. They find that tradeoffs exist between power and performance and describe how software could considerably decrease its power consumption while only marginally affecting performance, by making software aware of its power consumption.

We take a similar approach regarding the direct comparison of energy and run time without a notion of efficiency and also measure complex software systems.

While Tsirogiannis et al. consider software configurations to some extent, none of these studies explicitly consider configurable systems or the influences of specific configuration options on the performance or energy consumption.

Performance Influences of Configurable Systems

Among others, Siegmund et al. and Jamshidi et al. investigate the performance of configurable software systems. In particular, they model the influences of specific configuration options and interactions between options on the performance of a system. Siegmund et al. [SGAK15] propose an approach for creating performance-influence models, that describe the influence of individual configuration options and interactions among them on performance or other non-functional properties. Jamshidi et al. [JSV$^+$17] apply transfer learning to these models to avoid or reduce the cost of relearning models by adapting them to environmental changes.

Performance and Energy in Configurable Systems

In previous work [Wer17], we combined the previous two approaches, of on the one hand, comparing performance and energy consumption, and on the other hand, investigating performance influences of configuration options in configurable software systems. We investigated influences of configuration options on both the performance and energy consumption and compared performance and energy consumption, as well as their correlation, for different configurations. We found that performance and energy consumption are generally linearly correlated, however, how strong the correlation is, depends on the type of software system. We identified two different types of software systems, application software and infrastructure software, with different correlation behaviour, namely that there is generally a stronger correlation between performance and energy consumption in application software. We found that the correlation behaviour also depends on specific configuration options. In particular, for infrastructure software, only specific configuration options result in the weaker correlation compared to application software.

In this thesis, we take a similar approach of comparing performance and energy consumption of configurable software systems, additionally considering the aspect of software evolution. We also consider again the correlation between performance and energy consumption and investigate whether the correlation changes over time.

Performance Evolution

The work of Sandoval Alcocer et al. [SBDD13] is concerned with the the evolution of performance, i.e. changes in performance caused by the evolution of software. Software evolves by being changed over time. They propose a visualization approach to aid in investigating performance evolution.

Mühlbauer et al. [MAS19] model the evolution of performance in different software systems, including configurable systems. They use machine learning to estimate the evolution of performance with a minimal measurement effort.

In his master's thesis, Hasreiter [Has19] combined two different aspects, one of them being performance influences in configurable software systems and the other one being performance evolution. In his exploratory study, he measured the performance for multiple configurations and releases of different software systems. He found that performance changes can be attributed to specific releases and in some cases also to specific configuration options. Additionally, he investigated whether these changes are also reflected in documentation.

We combine the approach of Hasreiter with that of our own previous work to investigate performance changes but also energy changes and the relation between the two, in configurable and evolving software systems, i.e. we consider both configurations and releases of a software system.

3. Background

In this chapter, we provide background information related to the topic of this thesis and introduce terminology used throughout the following chapters.

3.1 Types of Software

The related work in Chapter 2 indicates, that the behaviour of performance and energy consumption, and in particular their correlation is different depending on the type of software system.

Therefore, we continue to use the terminology we used in our previous work [Wer17] and distinguish between *application software* and *infrastructure software*.

Infrastructure software is typically used in a server–client scenario. It runs continuously, whether it is currently in use (e.g. serving requests) or not. This means, that resources are consumed whether the software is performing a task or not. Examples for this type of software include web servers and database servers.

Application software, on the other hand, only runs when it is actually in use. Resources are only used when the software performs a task; when there is no task to perform, the software does not run. Examples for this type of software include compression tools and video encoders.

3.2 Configurable Software Systems

Nowadays, most software is configurable. Being configurable means that there are *configuration options* (also called *features*), that make it possible to enable or disable specific functionality of the software or to change the behaviour of specific functionality. *Configuration options* can provide variability at compile-time or at runtime. We focus on *configuration options* that can be specified at runtime, e.g. as command line parameters or in a configuration file (with one exception of a *configuration option* that we need to specify at compile-time for compatibility reasons).

We distinguish between two different types of *configuration options*. *Binary options* can either be selected (enabled) or deselected (disabled). *Numeric options* have a specific real number value from a set of valid values for the particular option.

A specific valid combination of *configuration options* is a *configuration*. Not all combinations of *configuration options* are valid. There may be *constraints* on the set of valid combinations, e.g. a *configuration option* may be required or two *configuration options* might be mutually exclusive.

Configuration options and *configurations* can mathematically be represented as sets. We reuse the notation used by Siegmund et al. [SGAK15] and denote the set of all *configuration options* as $\mathcal{O}$ and the set of all *configurations* as $\mathcal{C}$. The set of all *configurations* is also called the *configuration space*. A *configuration* $c \in \mathcal{C}$ is a function $c : \mathcal{O} \to \mathbb{R}$. For a *binary option* $o_b \in \mathcal{O}$, $c(o_b) = 1$ if the option is selected in the configuration c, $c(o_b) = 0$ otherwise. For a numeric option o_n, $c(o_n) = x, x \in \mathbb{R}$ where x is the value for option o_n in the configuration c and x is a valid value for option o_n. *Constraints* on valid *configurations* are Boolean expressions.

In some situations we want to express a specific subset of the *configuration space*. We can define such a subset $\mathcal{C}_s \subset \mathcal{C}$ by a *constraint* s.

The size of the *configuration space* grows exponentially with the number of *configuration options* and is huge for many contemporary software systems as discussed by Xu et al. [XJF⁺15]. We use a *feature model*, consisting of the set of *configuration options*, together with a set of *constraints* to represent the *configuration space*.

A *feature model* can be graphically represented as a *feature diagram* which consists of a tree visualizing the *configuration options* with their *constraints* and additional so called *cross-tree constraints*, which are *constraints* that cannot be represented in the tree.

In Figure 3.1, we show an example for a *feature diagram* using our notation. The *feature diagram* visualizes a *feature model* describing *configurations* of the software 'ExampleDatabase'. The set of *configuration options* is:

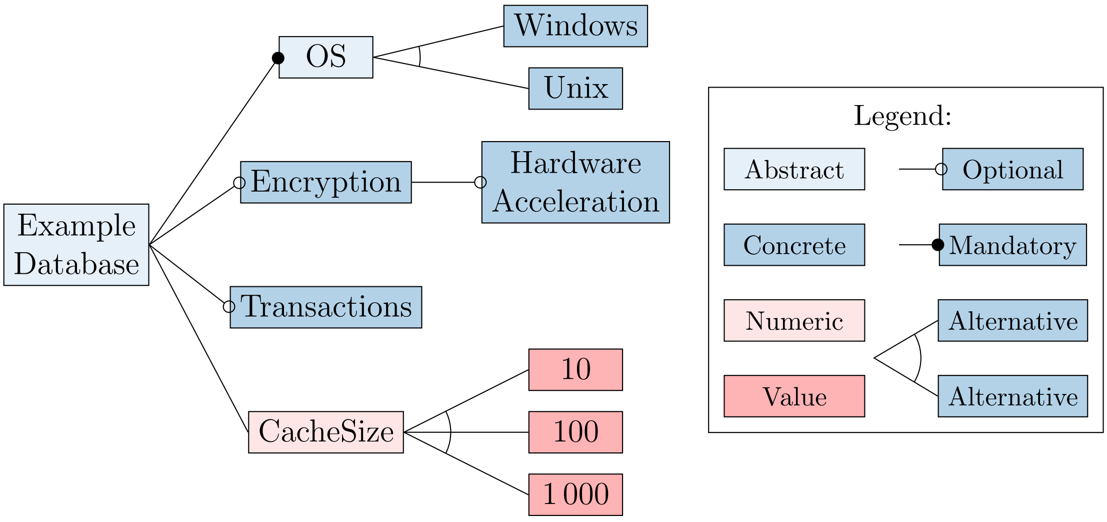
$$\begin{aligned}\mathcal{O} = \{&\mathrm{OS}, \mathrm{Windows}, \mathrm{Unix}, \mathrm{Encryption}, \mathrm{HardwareAcceleration},\\ &\mathrm{Transactions}, \mathrm{CacheSize}\}.\end{aligned}$$

'OS' is a mandatory option, denoted by a filled circle, i.e. it has to be present in every *configuration*. This is equivalent to the *constraint* simply consisting of the variable OS.

'Transaction', 'Encryption' and 'HardwareAcceleration' are optional, denoted by an empty circle. Since 'HardwareAcceleration' is a child of 'Encryption', it may only be present in *configurations* with 'Encryption'. This is equivalent to the following *constraint*:

$$\mathrm{HardwareAcceleration} \implies \mathrm{Encryption}$$

'Windows' and 'Unix' form an *alternative group*. One of these options is required (for configurations in which the parent option is present, which is always the case



Cross-tree constraints:
¬Encryption ∨ ¬Transactions

Figure 3.1: Example *feature diagram* for a database with binary *configuration options* ‘OS’, ‘Windows’, ‘Unix’, ‘Encryption’, ‘HardwareAcceleration’ and ‘Transactions’, and a numeric *configuration option* ‘CacheSize’.

in the example since ‘OS’ is mandatory), but they are mutually exclusive. This is equivalent to the following *constraints*:

$$\text{Windows} \lor \text{Unix} \implies \text{OS}$$
$$\neg\text{Windows} \lor \neg\text{Unix}$$

The option ‘CacheSize’ is numeric and has the set $\{10, 100, 1\,000\}$ of valid values.

In the example, ‘Encryption’ and ‘Transactions’ are mutually exclusive, which cannot be represented as part of the tree of the *feature diagram*. The corresponding *constraint* is simply included underneath the tree as a *cross-tree constraint*.

3.3 Software Evolution

According to Lehman [Leh80], a piece of software, once developed, is not static but there is a constant need for improvement. Software *evolves* over time. Some of the most common reasons for the need to modify existing software are the fixing of bugs, the mitigation of security issues or the addition of new functionality.

Nowadays, the source code for most software is stored in a version control system such as *Git*[1] or *Subversion*[2] (*SVN*) . In these systems, changes to the source code can be *committed* and assigned a descriptive message. Changes are typically only committed when a meaningful increment to the software has been implemented (e.g. one specific bug fixed). We call each of these small, meaningful increments of a software *revisions*.

[1] https://git-scm.com/ – last visited on 2019-08-18
[2] https://subversion.apache.org/ – last visited on 2019-08-18

In many cases, a *revision* is not considered to be a finished version of the software (e.g. new functionality is implemented in several small increments over the course of several *revisions*). Only some *revisions* are thus considered to form a new version of the software and are assigned a version number. These are then published, or *released*, by the maintainer of the software. We call such a *revision* a *release*.

In this thesis, we focus only on *releases*.

3.4 Performance

Performance can be defined and measured in different ways. The term is commonly used to refer to how well or how fast a system operates. How well a system performs, can be seen as either the amount or quality of the work performed.

Tsirogiannis et al. [THS10] define performance generically as a ratio of the amount or quality of work done to the time needed for it:

$$\text{Performance} = \frac{\text{Work done}}{\text{Time}} \tag{3.1}$$

In this thesis, we do not consider any qualitative attributes of the performed work and are instead only interested in the time it takes to perform a specific workload. Since the amount of *Work done* is now constant, performance is only defined by the time it takes to execute the workload. Thus, we can use a simplified definition of *performance*:

$$\text{Performance} = \text{Run time} \tag{3.2}$$

The different types of software we introduced in Section 3.1, *application software* and *infrastructure software*, require different approaches to measuring *performance*.

Application software directly executes the workload and runs exactly for the amount of time it needs to perform it. *Performance* can be measured directly by determining the execution time of the measured software.

Infrastructure software runs in a client–server setup where the measured software runs as server. The server may already be running before the workload is started and will remain running after it is completed. The workload itself is executed with the help of a benchmark client that communicates with the measured software over a network. The benchmark client runs only for the duration of the workload. Consequently, while *performance* is still determined by the execution time of the workload, it is no longer measured directly at the measured software but rather at the benchmark client.

Note, however, that while the *performance* is determined by the execution time of a benchmark, this does not necessarily mean that the time span from starting a process to its end is measured. In many cases, both *application software* and benchmark clients for *infrastructure software* write accurate timing information to a log file. These times can more accurately represent the execution time of the workload and we use them whenever they are available.

Throughout this thesis, we use the symbol p to denote a specific *performance*.

3.5 Energy Consumption

We measure the *energy consumption* of the whole system, rather than the energy consumption of individual components like the CPU.

Due to restrictions in the experimental setup, we cannot directly measure the energy consumption, but we can measure the power consumption periodically. Since energy is defined as the product of power and time, we can calculate the *energy consumption* as a sum of power consumption values over time:

$$E = \sum_{t=\left\lfloor \frac{t_{\text{start}}}{\Delta t} \right\rfloor}^{\left\lceil \frac{t_{\text{end}}}{\Delta t} \right\rceil} P(t\ \Delta t) \cdot \Delta t \tag{3.3}$$

E is the *energy consumption* between the start of the measurement t_{start} and the end of the measurement t_{end}. $P(t)$ is the power consumption measured at the time t and Δt is the interval between individual measurements of the power consumption.

Since we can only accurately measure *energy consumption* for time spans that are multiples of the measurement interval Δt, and the accuracy of the *energy consumption* generally depends on the accuracy of the power consumption measurements, we want to set the measurement interval as short as possible.

With our measurement setup, setting $\Delta t = 1s$ is reasonable. Since we want to determine the *energy consumption* in units of power consumption per second, this interval considerably simplifies the calculation for *energy consumption*. *Energy consumption* can simply be calculated as the sum of power consumption values for each second during the measurement:

$$E = \sum_{t=\lfloor t_{\text{start}} \rfloor}^{\lceil t_{\text{end}} \rceil} P(t) \cdot 1s \tag{3.4}$$

t_{start} and t_{end} are measured in seconds. E and $P(t)$ are used as above.

Just as the distinction between *application software* and *infrastructure software* affects the definition of *performance*, it also has an impact on the notion of *energy consumption*.

For *application software*, measuring *energy consumption* is straightforward. *Energy consumption* is simply measured and calculated according to Equation 3.4 from the start time until the end time of the execution of the workload.

Infrastructure software, on the other hand, behaves different from *application software* in that it runs even when it is not performing any tasks. While this is irrelevant for *performance*, it is of interest for *energy consumption*, since the system running the software consumes energy even when the software is idle. This raises the question of whether and how the idle *energy consumption* should be included in the measurement. While in a real world scenario, one could certainly argue to define the energy consumption of one period of active use to include the idle energy consumption before or after that period up to the next period of active use. However, this is not suitable in an experimental environment, where the duration of the idle period is not determined by the utilization of the *infrastructure software* but rather by the experimental setup which defines when a measurement begins and when it ends. Moreover, for some comparisons of *performance* and *energy consumption* it is desirable to include the idle *energy consumption* to emulate a real world scenario, whereas for other comparisons, using the same definition of *energy consumption* as for *application software* is preferable.

In this thesis, we answer the question by determining what we call the *fixed time energy consumption* and using this notion of *energy consumption* whenever appropriate. Conceptually, *fixed time energy consumption* is the *energy consumption* of *infrastructure software* during a fixed time span that is the same for all *configurations* of a case study. The fixed time span has to be large enough to contain the execution time of the slowest *configuration*. *Fixed time energy consumption* includes the *energy consumption* during active use and the idle *energy consumption* until the end of the *fixed time*.

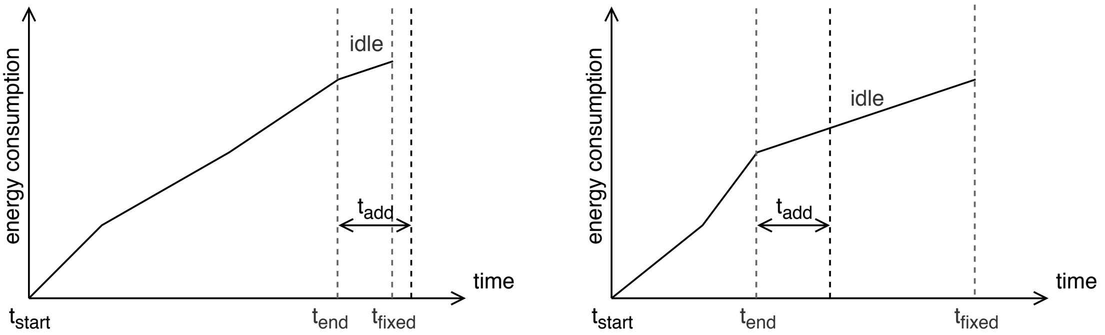


Figure 3.2: *Fixed time energy consumption* for *infrastructure software*. The left graph shows the *energy consumption* over the course of a measurement for a slow *configuration* and the right graph for a fast *configuration*. The *energy consumption* after the end of the benchmark t_{end} until the conceptual end of the measurement with a fixed duration t_{fixed} is the idle energy consumption of the system. The actual measurement starts at t_{start} and ends at $t_{\text{end}} + t_{\text{add}}$.

Since in many cases there is a large discrepancy in the execution time between the fastest and slowest *configurations* of a case study, it is not practical to measure all *configurations* for the whole duration of the *fixed time*. Instead, we assume that the idle *energy consumption* is sufficiently constant for each *configuration* to be extrapolated from a short period of time t_{add}. We define the extrapolated *fixed time energy consumption* E_{fixed} with *fixed time* t_{fixed} as follows:

$$E_{\text{fixed}} = E + \frac{t_{\text{fixed}} - t_{\text{end}}}{t_{\text{add}}} \cdot E_{\text{add}} \tag{3.5}$$

E is the *energy consumption* between t_{start} and t_{end} as defined above. E_{add} is analogously the *energy consumption* between t_{end} and $t_{\text{end}} + t_{\text{add}}$.

We illustrate the situation in Figure 3.2.

Throughout this thesis, we use the symbol e to denote a specific *energy consumption*. Since the *fixed time energy consumption* is only defined for *infrastructure software*, we use e^f to denote the *fixed time energy consumption* for *infrastructure software* but the (regular) *energy consumption* for *application software*.

3.6 Performance-Influence and Energy-Influence Models

Siegmund et al. [SGAK15] introduce the concept of a *performance-influence model* to describe the *performance* of configurable systems. More specifically, such a model describes the influence of individual *configuration options* and interactions between *configuration options* on the *performance* of all *configurations*.

Interactions of *configuration options* are influences that only appear with a certain combination of *configuration options*, but not with the individual *configuration options* by themselves. Interactions can have obvious reasons such as the existence of code that is only ever executed with a specific combination of *configuration options*. Sources of interactions can however be more subtle. For example, in a database system, a *configuration option* ‘Encryption’ might not change its behaviour whether another *configuration option* ‘Compression’ is selected or not, and will always execute the same code. However, when compression is enabled, the performance impact of the encryption might still change, simply because compression results in a reduction of the amount of data that needs to be encrypted.

While these models are introduced as *performance-influence models*, they can be used to describe influences of *configuration options* on any measurable *non-functional property*, in particular also the *energy consumption*. We use the term *performance-influence model* for a model of the *performance* and the term *energy-influence model* for a model of the *energy consumption*. When generically referring to a model of any *non-functional property*, we simply use the term *influence model*.

In addition to introducing *influence models*, Siegmund et al. [SGAK15] also propose an iterative machine-learning algorithm to derive such models from a sample of measured *configurations* (or the whole population) and provide an implementation of the algorithm with the tool *SPL Conqueror*[3]. We use *SPL Conqueror* to generate the *performance-influence and energy-influence models* for this thesis.

We generate *performance-influence and energy-influence models* for each *release* of each case study and use them to compare between *releases* and between *performance* and *energy consumption*. This requires us to have comparable models. While both binary and numeric *configuration options* can be represented in *influence models*, we choose to convert numeric options to binary options by *discretizing* them. Consequently, we only need to represent binary *configuration options* in *influence models*. To *discretize* a *feature model*, each numeric *configuration option* is replaced with an alternative group with one alternative for each valid value of the numeric option.

[3] https://github.com/se-passau/SPLConqueror – last visited on 2019-08-18

The general form of a *performance-influence model* is as follows:

$$\Pi(c) = \beta_0 + \sum_{o \in \mathcal{O}} \phi_o(c(o)) + \sum_{o_1..o_n \in \mathcal{O}} \Phi_{o_1..o_n}(c(o_1)..c(o_n)) \tag{3.6}$$

$\Pi(c)$ is the *performance* of a *configuration* $c \in \mathcal{C}$. The set of all *configuration options* $\mathcal{O}$ and the set of all *configurations* $\mathcal{C}$ are used as defined in Section 3.2. β_0 is a constant base value for the *performance* of all *configurations*, which is independent of the *configuration*. $\phi_o(c(o))$ is the influence on the *performance* of a single *configuration option* $o \in \mathcal{O}$ based on its presence in the *configuration* c and $\Phi_{o_1..o_n}(c(o_1)..c(o_n))$ is the influence on the *performance* from multiple *configuration options* $o_1..o_n \in \mathcal{O}$ based on their presence in the *configuration* c.

Analogously to the definition of a *performance-influence model* $\Pi(c)$, we denote an *energy-influence model* as $E(c)$.

The following example could be an excerpt from a *performance-influence model* for the example database system described in Section 3.2 with the *feature diagram* in Figure 3.1. Abbreviations are used for the *configuration options*: Transactions (T), Encryption (E), HardwareAcceleration (H), CacheSize=10 (C_{10}), CacheSize=1 000 ($C_{1\,000}$).

$$\begin{aligned} \Pi(c) = 100 + 20 \cdot c(T) + 50 \cdot c(E) - 30 \cdot c(E) \cdot c(H) \\ + 15 \cdot c(E) \cdot c(C_{10}) - 10 \cdot c(T) \cdot c(C_{1\,000}) \end{aligned} \tag{3.7}$$

In the example, there are interactions between ‘Encryption’ and ‘HardwareAcceleration‘, between ‘Encryption’ and a ‘CacheSize’ of 10, as well as between ‘Transactions’ and a ‘CacheSize’ of 1 000. From the model, the *performance* of a *configuration* could be calculated by simply plugging in a *configuration* c into the formula. For example, a *configuration* c_1 with ‘Encryption’, ‘Transactions’ and a ‘CacheSize’ of 10 would have the following performance according to the model:

$$\begin{aligned} \Pi(c_1) &= 100 + 20 \cdot c_1(T) + 50 \cdot c_1(E) - 30 \cdot c_1(E) \cdot c_1(H) \\ &\quad + 15 \cdot c_1(E) \cdot c_1(C_{10}) - 10 \cdot c_1(T) \cdot c_1(C_{1\,000}) \\ &= 100 + 20 \cdot 1 + 50 \cdot 1 - 30 \cdot 1 \cdot 0 + 15 \cdot 1 \cdot 1 - 10 \cdot 1 \cdot 0 \\ &= 100 + 20 + 50 - 0 + 15 - 0 = 185 \end{aligned} \tag{3.8}$$

When comparing *influence models*, it is desirable to have the same terms, i.e. *configuration options* and interactions, in all models. This is impractical with the default learning approach of *SPL Conqueror*, which is iterative: An optimal term is added in every iteration until the algorithm terminates, either once a certain error rate has been reached or when an iteration no longer yields an improvement that exceeds a specified threshold. Consequently, models for different datasets that have a common set of *configuration options* will usually not all contain the same terms.

SPL Conqueror offers an alternative to the iterative approach: a fitting algorithm that takes model terms as input and determines only the factors for each given term.

We combine these two approaches by using the iterative algorithm to determine relevant terms and then manually combine and filter those terms from different models. We then use the resulting set of terms to generate all models using the fitting algorithm.

3.7 Pearson Correlation

Since we are examining the correlation between *performance* and *energy consumption*, we need a metric for the correlation. We use the *Pearson correlation coefficient*, which was developed by Pearson towards the end of the 19th century [Pea96]. It is a measure for the linear correlation between two variables. Benesty et al. [BCHC09] define the *Pearson correlation coefficient* as follows:

$$\begin{aligned} \rho(a,b) &= \frac{cov(a,b)}{\sigma_a \sigma_b} \\ cov(a,b) &= \frac{1}{n}\sum_{i=1}^{n}(a_i - \bar{a})(b_i - \bar{b}) \\ \sigma_a &= \frac{1}{n}\sum_{i=1}^{n}(a_i - \bar{a})^2 \end{aligned} \tag{3.9}$$

$\rho(a,b)$ is the *Pearson correlation coefficient* of a and b, $cov(a,b)$ is the cross-correlation, or covariance, between a and b, and σ_a is the standard deviation of a. $\bar{a}$ is the mean of a.

The *Pearson correlation coefficient* can assume values ranging from -1 to 1. A *Pearson correlation coefficient* of 0 indicates no linear correlation, while a *Pearson correlation coefficient* with an absolute value approaching 1 indicates a strong linear correlation. An absolute value of less than 0.5 is generally said to indicate only a weak linear correlation. Absolute values greater than 0.8 typically indicate a strong correlation.

4. Methodology

In this chapter, we describe the methodology we use in this thesis. First, we introduce our research questions. Next, we list the case studies we use for our experiments. It follows our experimental setup and we conclude the chapter with a more in-depth description of the operationalization of our case studies.

4.1 Research Questions

We have two research questions which are subdivided into respectively three and two subordinate questions with different levels of abstraction. In the first research question we investigate the evolutionary behaviour of *performance* and *energy consumption* independently and compare between the two. In the second research question, we investigate the *correlation* between *performance* and *energy consumption* and how it changes between different *releases*.

4.1.1 RQ1: Performance and Energy Consumption

In our first research question, we consider the two *non-functional properties performance* and *energy consumption* independently and compare between them. We investigate if there are changes in the *performance* and *energy consumption* of software systems across *releases*, i.e. over time, and whether these changes behave similarly for *performance* and *energy consumption*.

> **RQ1:** Are there changes in the *performance* and *energy consumption* across releases?

RQ1.1: Are there changes in the mean performance and mean energy consumption across releases?

In the first subordinate question, we do not consider individual *configurations* but rather use the mean of the *performance* and *energy consumption* over all *configurations* for each *release*. We investigate both the quantity and quality of changes in *performance* and *energy consumption* between consecutive *releases*. Additionally, we compare whether these results are the same or different for *performance* and *energy consumption*. Using the mean values allows us to get an overview on the evolutionary behaviour of *performance* and *energy consumption* in a simple manner, without having to consider a value for every single *configuration*.

RQ1.2: Are there changes in the performance and energy consumption of individual configurations across releases?

Increasing the granularity, we now consider individual *configurations* in the second subordinate question. Again, we consider the quantity and quality of changes in *performance* and *energy consumption* between consecutive *releases* and whether there are differences in our observations between *performance* and *energy consumption*. The increased granularity (individual *configurations* rather than mean over all *configurations*) allows us to detect changes that may not affect all *configurations* or even opposing changes that may cancel each other out and may not appear in the mean values of the previous research question at all.

RQ1.3: Are changes in performance and energy consumption caused by specific individual features or feature interactions?

In a third step, we further increase the level of abstraction by investigating *configuration options* instead of *configurations*. To achieve this, we apply the learning algorithms of *SPL Conqueror* to generate *performance-influence and energy-influence models*. Now we are no longer investigating individual *configurations* but rather the influence of specific *configuration options* and combinations of *configuration options*. This allows us to relate specific changes in the *performance* and *energy consumption* to specific *configuration options* and interactions.

4.1.2 RQ2: Correlation between Performance and Energy

In the second research question, we consider the *correlation* between *performance* and *energy consumption*. We investigate whether the *correlation* between *performance* and *energy consumption* changes across *releases*.

RQ2: Are there changes in the *correlation* between *performance* and *energy consumption* across releases?

RQ2.1: Are there changes in the *correlation* between performance and energy consumption across releases?

In the first subordinate question concerned with the *correlation*, we compare the *correlation* between *performance* and *energy consumption* of consecutive *releases*. In this question, we consider the *correlation* of all *configurations* for each *release*.

RQ2.2: Are changes in the *correlation* between performance and energy consumption caused by specific individual features?

For this subordinate question, we increase the granularity and consider the *correlation* between *performance* and *energy consumption* for different subsets of the *configuration space*. We take a subset of the *configuration space* for each *configuration option* and for each valid value of the respective *configuration option*. We then compare whether changes in the *correlation* across consecutive *releases* are different depending on the *configuration option*. This allows us to abstract from the *configurations* and investigate the *correlation* behaviour of the different *configuration options*.

4.2 Case Studies

In the previous section we presented our research questions. To answer these questions, we measure the *performance* and *energy consumption* of four case studies and evaluate the results. In this section, we describe the case studies. Three of the case studies, *HSQLDB*, *Apache httpd* and *PostgreSQL* are *infrastructure software*. The fourth case study, *libvpx VP8* is *application software*. For each of the case studies, we present a *feature model* to describe the *configuration space*, explain which *releases* we selected and describe the workload or benchmark we use.

4.2.1 HSQLDB

Our first case study, *HyperSQL DataBase*[1], or *HSQLDB*, is a SQL database written in Java. It can be embedded into applications or run as stand-alone database server. We are using it as database server, which makes it an example for *infrastructure software*.

Configuration Space

We used the documentation[2] of *HSQLDB* to identify *configuration options* and selected suitable *configuration options* based on previous experience with measuring *HSQLDB*.

In the following list, we describe the *configuration options* and in Figure 4.1 we show the corresponding *feature model*.

memoryTables: By default, *HSQLDB* stores database tables completely in memory. This *configuration option* indicates the default behaviour.

cachedTables: This *configuration option* configures *HSQLDB* to store database tables on disk and only keep some of the data in memory.

cacheSize: This numeric *configuration option* configures the amount of data to keep in memory for cached tables. The unit is kilobytes. The default value is 10 000.

[1] http://hsqldb.org/ – last visited on 2019-08-23

[2] http://hsqldb.org/doc/2.0/guide/guide.pdf – last visited on 2019-08-23

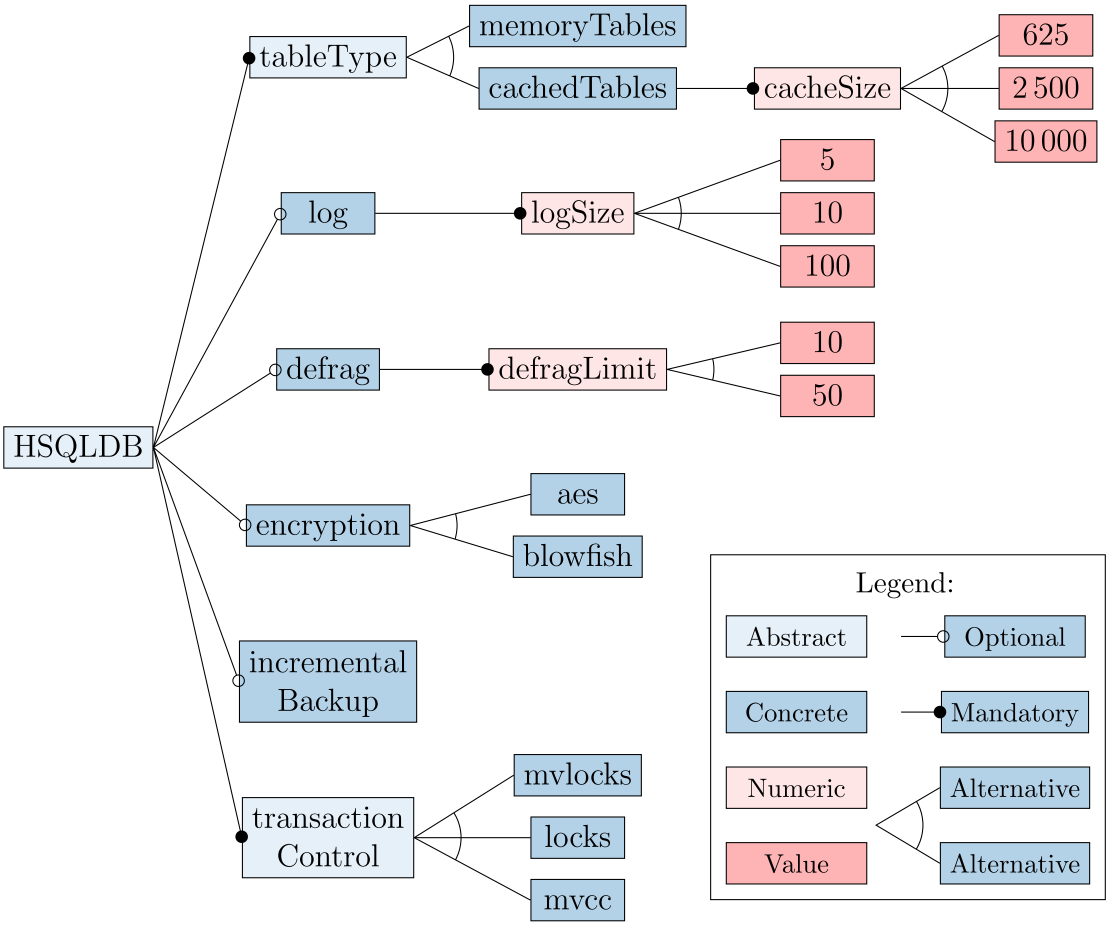


Figure 4.1: *Feature diagram* for the *HSQLDB* case study

log: This *configuration option* specifies whether *HSQLDB* should write data changes to a file which could be used to recover from an abrupt shutdown. Logging is enabled by default.

logSize: This numeric *configuration option* configures the maximum size of the log file used with the **log** option. When the file reaches the limit (in megabytes), *HSQLDB* performs a *checkpoint* which updates the persistent storage and clears the log file. We did not include the default value of 50 to be able to use a wider range of values, and previous experience with *HSQLDB* has shown us that lower values are more likely to affect the *performance* and *energy consumption*.

defrag: This *configuration option* specifies whether *HSQLDB* should defragment the persistent storage, i.e. rewrite the persistent storage file to free unused space. By default, defragmentation is not performed automatically.

defragLimit: This numeric *configuration option* specifies a percentage of wasted space that needs to be present in the persistent storage file before defragmentation is performed.

encyption: This *configuration option* specifies whether *HSQLDB* should encrypt the database.

aes, blowfish: These *configuration options* specify whether to use AES or Blowfish for database encryption. ECB mode is used in both cases, since specifying an IV is only supported in *HSQLDB* since *release* 2.4.1. No encryption is used by default.

incrementalBackup: This *configuration option* specifies whether to update the persistent storage incrementally during operation, rather than all at once during a checkpoint. Incremental backups are enabled by default.

locks, mvlocks, mvcc: These *configuration options* specify which transaction control mode to use. The default mode is LOCKS. The other modes are MV-LOCKS and MVCC.

This *feature model* represents a total number of 864 *configurations*.

Releases

The source code of *HSQLDB* is available in an SVN repository[3]. Each *release* is indicated with a *tag* labelled with a version number.

The first *release* we measured was 2.1.0, which was released in March of 2011. While there have been *releases* before 2.1.0, we would have needed to use Java version 6 or earlier to compile and run them. All *releases* starting with 2.1.0 can be compiled and run with Java 8. There are two reasons why we did not use older *releases* of *HSQLDB*. Firstly, it is difficult to acquire and run Java JDK 6 on a current operating system. Secondly, we did not want to use different version of the Java runtime to run different *releases* since this could lead to unexpected influences on the *performance* and *energy consumption* arising from differences in the Java version rather than changes in *HSQLDB*.

The last *release* we measured was 2.4.1, which was released in May of 2018. At the time of this writing, version 2.5 is available, but it had not yet been released when we performed the measurements for this case study.

We did not leave out any releases between 2.1.0 and 2.4.1, resulting in a total count of 19 *releases* of *HSQLDB*, spanning seven years.

A complete list of *releases* for this case study is included in Section A.2 of the appendix.

Workload

We used the database benchmark *PolePosition*[4] in version 0.6.0 to generate load on the database server. *PolePosition* offers a number of different scenarios, including both simple and complex queries. We did not consider concurrent scenarios, because they require specifying a fixed duration which is incompatible with our notion of *performance* as execution time. Out of the remaining scenarios, we used all those

[3] https://sourceforge.net/p/hsqldb/svn/ – last visited on 2019-08-23

[4] http://www.polepos.org/ – last visited on 2019-08-23

that are compatible with JDBC, which is used to communicate with the *HSQLDB* server.

PolePosition offers parameters for each of the scenarios to adjust the number of iterations and amount of data involved. We chose values for these parameters experimentally, such that for each of the *configurations*, none of the scenarios would run too long and none of the scenarios would complete within only a few seconds. This is to ensure that measurements complete within the available time and at the same time that all scenarios are represented in the results for each *configuration*.

We used two instances of *PolePosition* with identical configuration to increase the load on the server.

4.2.2 Apache httpd

Our second case study, *Apache HTTP Server*[5], or *Apache httpd*, is a popular open-source web server written in C. Like the previous case study, it is another example for *infrastructure software*.

Configuration Space

As with the previous case study, we used the documentation[6] of *Apache httpd* to identify *configuration options* and selected suitable *configuration options* based on previous experience with measuring *Apache httpd*.

In the following list, we describe the *configuration options* and *cross-tree constraints*, and in Figure 4.1 we show the corresponding *feature model*.

	maxClients							
	512		1024		2048		4096	
ratio option	P	T	P	T	P	T	P	T
moreProcesses	128	4	128	8	256	8	256	16
evenThreadRatio	16	32	32	32	32	64	64	64
moreThreads	4	128	8	128	8	256	16	256

Table 4.1: Process (P) and thread (T) counts for the *Apache httpd* case study

prefork, worker: *Apache httpd* uses a *multi-processing module* (MPM) to accept connections and to create child processes or threads to handle requests. The *configuration options* **prefork** and **worker** select the MPM of the same name. The fundamental difference between the prefork and worker MPMs is that the prefork MPM uses only processes while the worker MPM uses both processes and threads.

[5] http://httpd.apache.org/ – last visited on 2019-08-23

[6] http://httpd.apache.org/docs/current/en/ – last visited on 2019-08-23

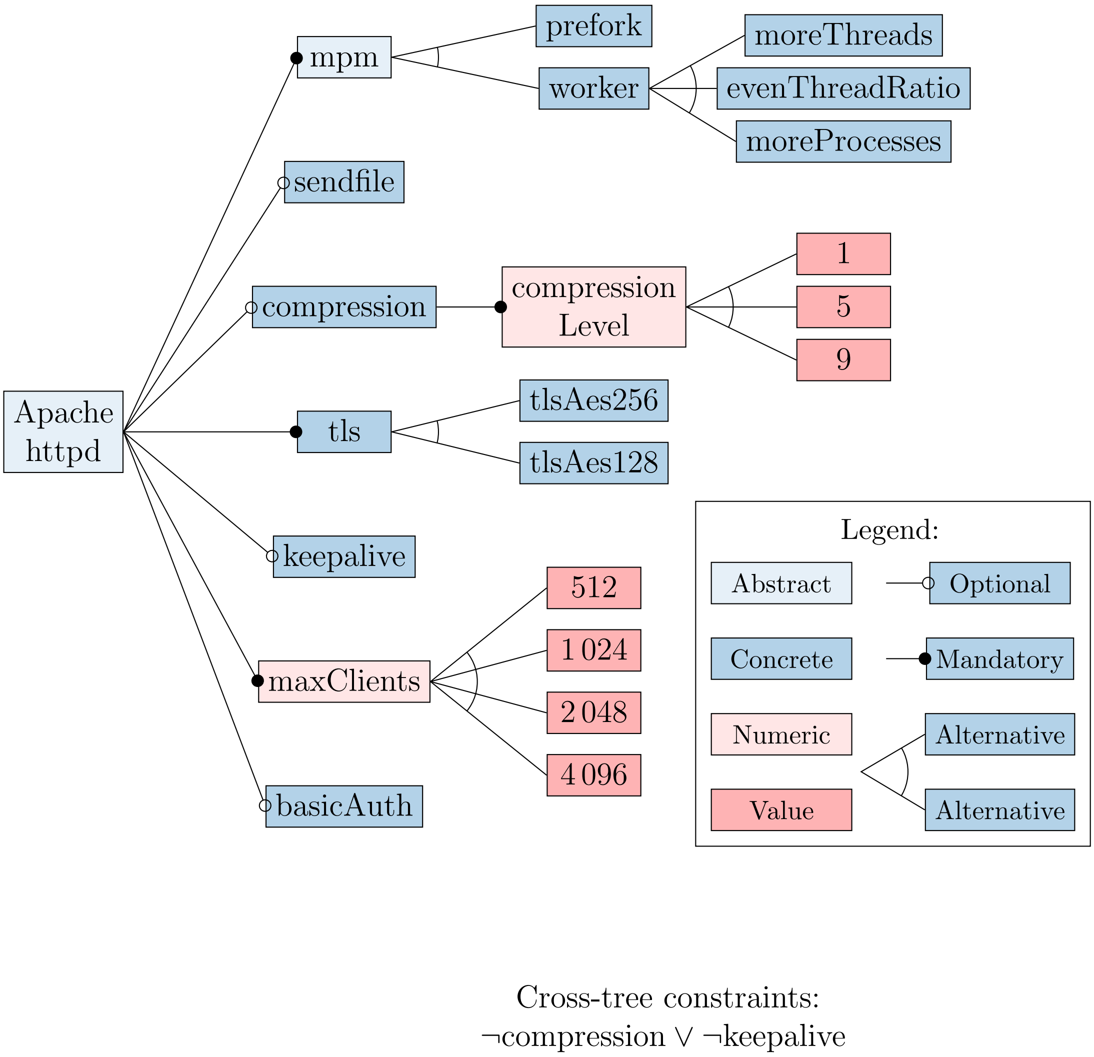


Figure 4.2: *Feature diagram* for the *Apache httpd* case study

moreThreads, evenThreadRatio, moreProcesses: With the worker multi-processing module, *Apache httpd* uses multiple processes and multiple threads within each process. These *configuration options* configure the ratio between processes and threads per process. Table 4.1 lists the number of processes and threads for each of these options.

sendfile: This *configuration option* enables *Apache httpd* to use the operating system's *sendfile* function to directly send static files over the network without the need to read the file into a buffer first and then send the file from that buffer.

compression: This *configuration option* specifies whether to compress responses with GZip.

compressionLevel: This numeric *configuration option* specifies the compression level to use for GZip compression.

tls: This mandatory option represents the use of a TLS encrypted connection. We only measured *configurations* with TLS encryption for the simple reason that without TLS encryption, many *configurations* are so fast that the limiting factor is the network between server and client, and we wanted to measure the performance of the server and not of the network.

tlsAes265, tlsAes128: These *configuration options* indicate which cipher suite is used for TLS, DHE-RSA-AES256-SHA or DHE-RSA-AES128-SHA, respectively.

keepalive: This *configuration option* enables *Apache httpd* to use persistent HTTP connections that can be used to serve multiple requests. Connections are reused for up to five requests.

maxClients: This numeric *configuration option* specifies the maximum number of requests that *Apache httpd* can handle at the same time. For the prefork MPM, this is the number of processes. For the worker MPM, this is the product of the numbers of processes and threads. Table 4.1 lists the number of processes and threads for each value of this option.

basicAuth: This *configuration option* specifies the use of HTTP basic authentication.

$\neg$*compression* $\vee$ $\neg$*keepalive*: This *cross-tree constraint* is necessary due to a restriction of HTTP 1.0, which our benchmark client uses. HTTP 1.0 requires responses to keepalive requests to include a response length in the header, which is not known ahead of time with compression.

This *feature model* represents a total number of 640 *configurations*.

Releases

The source code of *Apache httpd* is available in an SVN repository. There is also a Git mirror[7] available, which we used.

While there are tags with version numbers in the source repository, not all of these tagged *revisions* have actually been released. Since we focus only on *releases*, we used the official list of releases which is included in the source repository.

Due to difficulties in compiling and running *release* prior to 2.2.0, we decided not to consider older versions, leaving us with the ranges 2.2.*, 2.3.* and 2.4.* of *releases*. However, while some 2.3.* versions are listed as having been released, these are preview versions for 2.4 and thus do not fit our definition of *releases*.

We chose not to measure *releases* from the 2.2.* range that are newer than the oldest 2.4.* to achieve a more even distribution of *releases* over time. From the then available ranges 2.2.0 to 2.2.22 and 2.4.1 to 2.4.38, we chose 21 *releases* spread out as evenly across the covered time span of approximately 13 years as possible, leaving

[7] https://github.com/apache/httpd – last visited on 2019-08-23

us with 2.2.0 from December 2005 as first *release* and 2.4.38 from January 2019 as last *releases*.

A complete list of *releases* for this case study is included in Section A.2 of the appendix.

Workload

We used *ApacheBench*, or *ab*, which is a part of *Apache httpd*, to generate load on and measure the performance of the server.

We configured *ab* to send a total of 100 000 requests with 1 000 concurrent requests. Since under this kind of load, the server is not always able to answer requests, we accepted up to 2.5% failing requests.

Requests were sent for a single static HTML file of size 2 kilobytes, which we experimentally determined to be a good compromise between high network utilization and having to send more requests than the server can handle.

4.2.3 PostgreSQL

Our third case study, *PostgreSQL*[8], is an SQL database written in C. It is our third instance of *infrastructure software*.

Configuration Space

As with the previous case studies, we used the documentation[9] of *PostgreSQL* to identify *configuration options*. Since we wanted to use a wide range of *releases* and use only *configuration options* that can reasonably be assumed to impact *performance* and that are relevant with our workload at all, only few *configuration options* were available for us. We experimentally determined which of those *configuration options* do not have any impact on *performance* in our scenario by measuring some *configurations*. We excluded those *configuration options* from consideration and used all remaining *configuration options* for this case study.

In the following list, we describe the *configuration options* and in Figure 4.3 we show the corresponding *feature model*.

fullPageWrites: This *configuration option* specifies that *PostgreSQL* should under certain conditions write full memory pages to disk instead of only modified portions. This ensures that the database can recover from a crash. This option is enabled by default.

sharedBuffers: This numeric *configuration option* specifies the amount of memory (in megabytes) available for shared buffers. The default value is 128.

synchronousCommit: This *configuration option* specifies whether a transaction has to wait until all data is physically written to disk before it is reported as successful. Synchronous commits are enabled by default.

[8] https://www.postgresql.org/ – last visited on 2019-08-23

[9] https://www.postgresql.org/docs/manuals/ – last visited on 2019-08-23

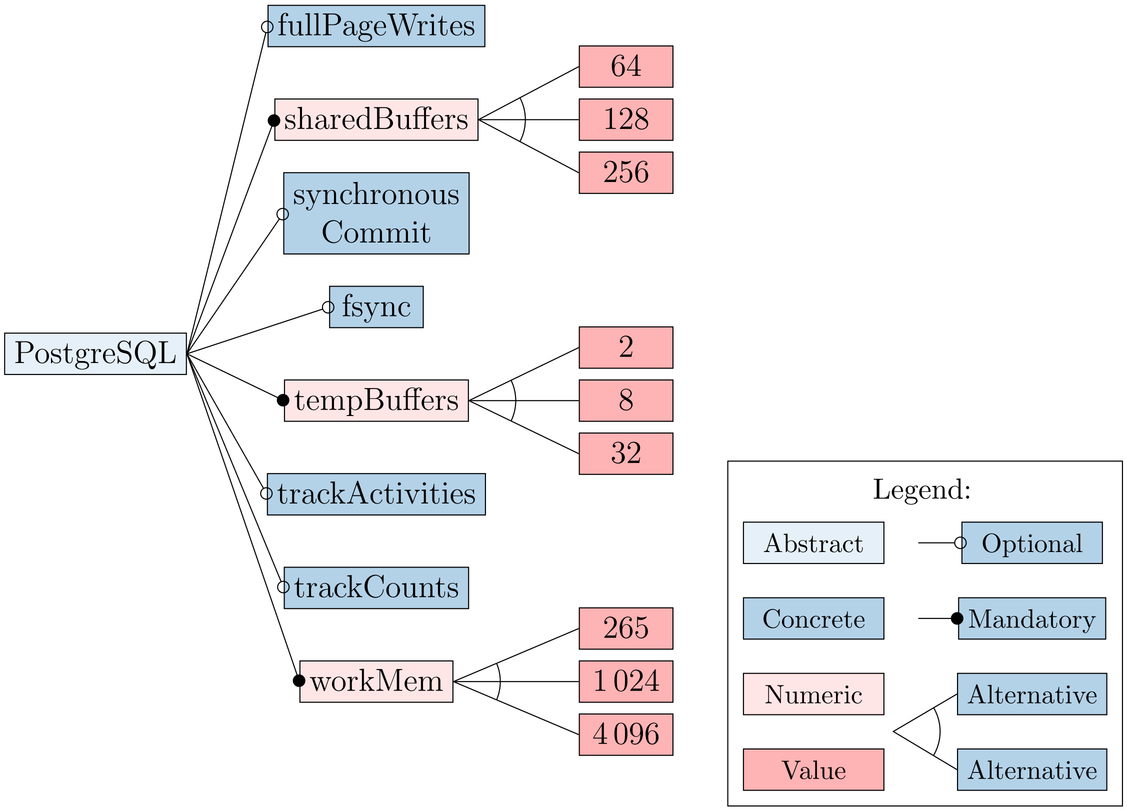


Figure 4.3: *Feature diagram* for the *PostgreSQL* case study

fsync: This *configuration option* specifies whether *PostgreSQL* should ensure that updates are physically written to disk, allowing the database server to recover from crashes. This option is enabled by default.

tempBuffers: This numeric *configuration option* specifies the amount of memory (in megabytes) available for temporary tables within each database session. The default value is 8.

trackActivities: This *configuration option* allows *PostgreSQL* to gather information on commands and timing information. This option is enabled by default.

trackCounts: This *configuration option* allows *PostgreSQL* to gather statistics on database activity and table sizes, which is required for certain optimization. This option is enabled by default.

workMem: This numeric *configuration option* specifies the amount of memory (in kilobytes) available for sorting and hashing operations. If these operations require more space, data is written to the disk. The default value is 4 096.

This *feature model* represents a total number of 864 *configurations*.

Releases

The source code of *PostgreSQL* is available in a Git repository[10]. We used the tags to identify *releases*.

[10] https://github.com/postgres/postgres – last visited on 2019-08-23

A new major version of *PostgreSQL* is released approximately once a year. Major versions receive updates in the form of minor *releases* for five years. This means, that at any time, approximately five different major versions of *PostgreSQL* are officially supported and receive updates[11].

Similar to the *Apache httpd* case study, we decided not do use *releases* from a major version that are newer than the next major version. In this way, the ranges of *releases* we measured for different major version do not overlap.

The first *release* we measured was 8.3.0, the initial *release* of the 8.3.* major version, which was released in February of 2008. We did not use any *releases* before that, since three of the *configuration options* we used, *synchronousCommits*, *trackActivities* and *trackCounts*, were only introduced in 8.3.0.

The last *release* we measured was 11.2 from the 11.* major version, which was released in February of 2019. At the time of the measurements, this was the most recent *release* of *PostgreSQL*.

From each of the major versions out of the range 8.3.* to 11.*, we used two *releases*, including the first one, for an even distribution of all releases over the covered time spam. This results in a total number of 22 *releases* of *PostgreSQL*, spanning eleven years.

A complete list of *releases* for this case study is included in Section A.2 of the appendix.

Workload

For the *PostgreSQL* case study, we used the same benchmark as in the *Apache httpd* case study, *PolePosition* 0.6.0. Since JDBC is used to communicate to *PostgreSQL* as well, we also used the same scenarios of *PolePosition*. We also used the same parameters for the different scenarios that we used for the *HSQLDB* case study.

We used two instances of *PolePosition* with identical configuration to increase the load on the server.

4.2.4 libvxp VP8

Our fourth and final case study is the VP8 encoder of *libvpx*[12], a video encoder for the WebM format, written in C. It is our only *application software* case study.

Configuration Space

We used the documentation of *libvpx VP8*[13] to identify *configuration options* and selected suitable *configuration options* based on the documentation and the results of Hasreiter [Has19].

In the following list, we describe the *configuration options* and in Figure 4.4 we show the corresponding *feature model*.

[11] https://www.postgresql.org/support/versioning/ – last visited on 2019-08-23

[12] https://www.webmproject.org/code/ – last visited on 2019-08-23

[13] https://www.webmproject.org/docs/encoder-parameters/ – last visited on 2019-08-23

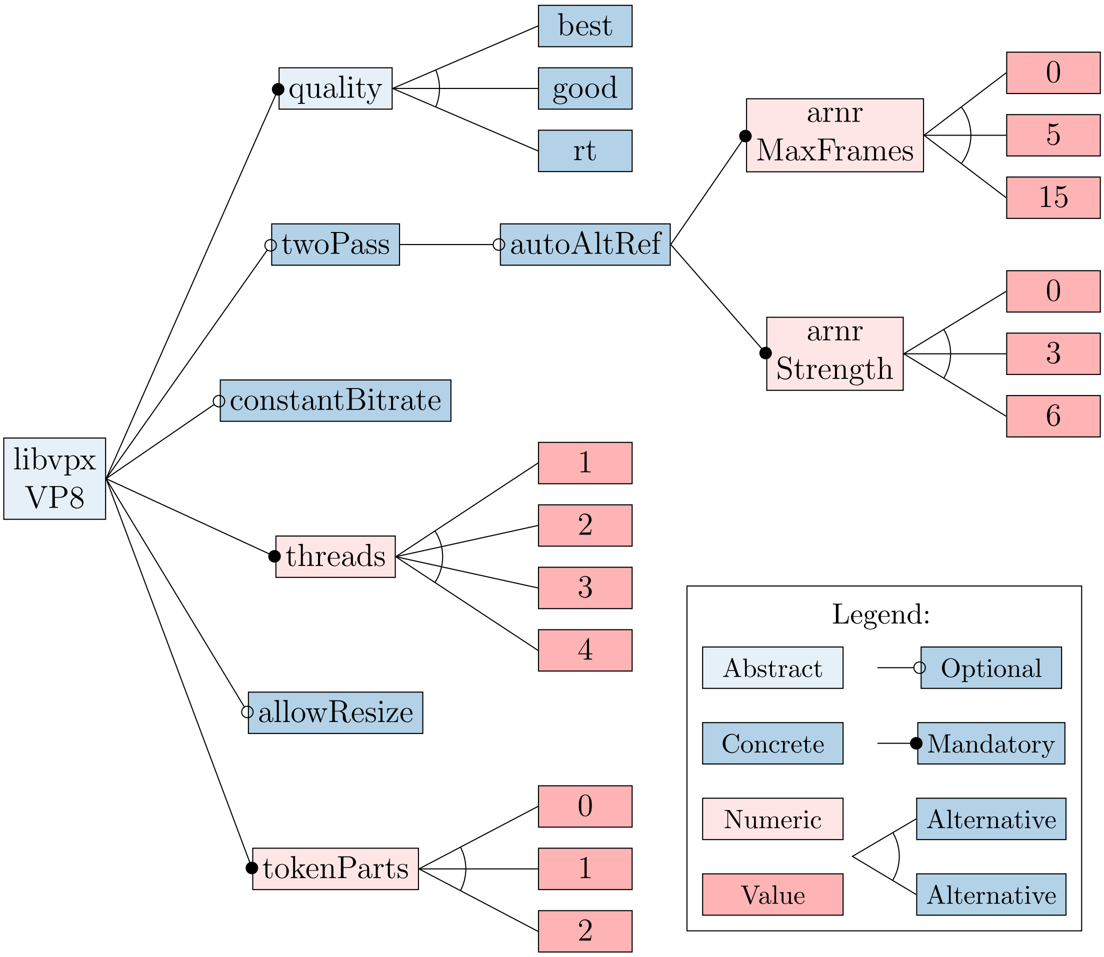


Figure 4.4: *Feature diagram* for the *libvpx VP8* case study

best, good, rt: These three *configuration options* specify if the encoder should focus on high output quality or speed. With **best**, *libvpx VP8* produces the best quality output at the expense of time. With **rt** (real-time), the encoder focuses only on speed and the quality depends on the available resources. The option **good** is the compromise between quality and speed.

twoPass: This *configuration option* controls whether *libvpx VP8* should encode in two passes. The first pass of two-pass encoding only gathers statistics for use in the second pass. This allows the second pass to use information about future frames. When two-pass encoding is not used, only a single encoding pass is performed which does not have access to statistics about subsequent frames.

autoAltRef: This *configuration option* allows the encoder to use a constructed (or alternate) reference frame[14] which is included in the output but is not a displayable frame. This reference frame may be derived from one or more future frames and is consequently only available with two-pass encoding.

[14] https://groups.google.com/a/webmproject.org/forum/#!topic/codec-devel/LUYpX2MUXgc – last visited on 2019-08-23

arnrMaxFrames: This numeric *configuration option* controls the behaviour of *libvpx VP8* when constructed reference frames are used. The value specifies the maximum number of future frames that may be used to derive a reference frame.

arnrStrength: This numeric *configuration option* controls the behaviour of *libvpx VP8* when constructed reference frames are used. The value specifies the level of noise filtering that is applied during the construction of reference frames.

constantBitrate: This *configuration option* specifies that the encoder should attempt to encode frames with a constant bitrate.

threads: This numeric *configuration option* specifies the number of threads used to encode portions of a frame in parallel. Not all steps of the encoding process can be parallelized.

allowResize: This *configuration option* controls whether the encoder is allowed to downscale frames during encoding.

tokenParts: This numeric *configuration option* specifies the logarithm (base two) of the number of partitions to use during encoding. Using more than one partition (value 0) allows more steps of the encoding process to be parallelized when using multiple threads.

We did not specify the default values for the listed *configuration options* since they are not specified in the documentation and may be different depending on the *release*.

This *feature model* represents a total number of 2 736 *configurations*.

Releases

The source code for *libvpx* is available in a Git repository[15]. All *releases* are marked with tags. When we performed the measurements, there were only 17 *releases*. We did not use the first *release*, v0.9.0, since it did not produce correct output files. We were also unable to use v1.6.0 due to an incompatibilty with the experimental setup. We used all the remaining 15 *releases* for this case study.

The first of our *releases* was v0.9.1 from June 2010. The last of our *releases* was v1.8.0 from February 2019, for a range of nine years.

Workload

Since *libvpx VP8* is *application software*, we do not need an additional client application to measure the *performance*. We can simply execute a specific workload with *libvpx VP8* and measure the execution time. Our workload was encoding an uncompressed video file to VP8 WebM (*.webm file format). As input video file, we used the 'Sintel' trailer[16] in YUV4MPEG2 (*.y4m file format) format with 480p resolution.

[15] https://chromium.googlesource.com/webm/libvpx – last visited on 2019-08-23

[16] https://media.xiph.org/ – last visited on 2019-08-23

4.3 Experimental Setup

In this chapter, we describe the experimental setup that we used to measure the case studies. We first describe the general hardware and measurement setup and then specify details that are relevant for the individual case studies.

4.3.1 Hardware and Measurement Setup

We measured all of our case studies on a cluster of workstation PCs. The cluster consists of 14 nodes with *Intel Core i5-4590* processors. The processors have four cores with a single thread per core and a base frequency of 3.3GHz. The nodes have 16GB of RAM and use SSDs for storage.

All nodes have a 1Gb/s *Intel Ethernet Connection I217-LM* and are connected to a single *Superstack 4 5500G* switch with 48 ports.

A minimal installation of Debian 9.9 is used as operating system.

All nodes are connected to power distribution units (PDUs), which measure the power consumption, independently for all the connected nodes. Twelve of the nodes are connected to an *IPT iPower P1* with 18 sockets and the remaining two nodes are connected to another *IPT iPower P1* with three sockets.

The PDUs measure the power consumption with a frequency of 1Hz. To log these power consumption values for later evaluation, we need to query these values over the network. Since querying and processing the data can introduce a small delay, we may end up with seconds without a value when querying exactly once every second. Since our definition of *energy consumption* (Section 3.5), where we sum up the power consumption values to directly obtain the *energy consumption*, relies on us having exactly one power consumption value for every second, we need to make sure that we do not miss a value for any second. For this purpose, we query power consumption values at a slightly higher rate of one value every 900 milliseconds, and simply discard a value when we have multiple values for the same second.

Since this client, that queries the power consumption values, has some power consumption of its own, we run it on a different host than the measured software so it cannot influence the results. Likewise, for case studies that consist of a server and client, we run the two on different nodes of the cluster, so that the client cannot impact the *performance* and *energy consumption* of the server.

4.3.2 Case Studies

In this section, we outline specific properties of our case studies that are relevant with our particular experimental setup.

HSQLDB

Since *HSQLDB* is written in Java and runs in the Java Virtual Machine (JVM), just-in-time (JIT) compilation can substantially impact the *performance* as stated by Georges et al. [GBE07]. Since we are not interested in measuring the start-up performance, we include a warm-up phase before the measurements for the *HSQLDB* case study. The warm-up phase uses the same server instance that is used for the benchmark but uses a separate database. While we are using two clients for the actual benchmark, we are only using a single client for the warm-up phase. To make sure that, during the warm-up phase, all code is executed on the server that is later used during the benchmark, so that it is already JIT compiled during the benchmark, we use a similar configuration for the client, *PolePosition*, for the warm-up phase as we use for the benchmark, just with a reduced number of iteration to save time.

We compiled all of our selected *releases* of *HSQLDB* as documented.

For this case study, we did not directly measure the *performance* as the execution time of the client, *PolePosition*, but instead used the run time reported by *PolePosition* in its log output.

Apache httpd

In the *Apache httpd* case study, the benchmark client *ab* sends a large number of requests to the server and needs to open a large number of sockets. Since, on Linux, sockets count towards the limit of open file handles, we had to raise this limit on both client and server.

To build older *releases* of *Apache httpd* on our modern systems, we had to deviate from the documented build steps. The tool *libtool* is used during the build process. We could use the current version 2 of *libtool* for versions 2.4.* of *Apache httpd* but had to use *libtool 1* for versions 2.2.*. *Apache httpd* has a dependency on *OpenSSL*. Older versions of *Apache httpd* are incompatible with new versions of *OpenSSL*. However, we found that the rather old version 0.9.8zh of *OpenSSL* was compatible with all of our *Apache httpd releases*, so we used this version for all *releases* to prevent different *OpenSSL* versions from impacting the *performance*. *Apache httpd* depends on the two libraries *APR* and *APR-Util*, which are also developed by *Apache*. Since we were not able to use the same versions of these libraries for all versions of *Apache httpd*, and we could not find documentation stating which version of *APR* and *APR-Util* should be used for which version of *Apache httpd*, we decided to use for every *Apache httpd* version, the versions of these libraries that had been the latest ones at the time the *Apache httpd* version was released.

Older versions of *Apache httpd* need to be compiled for a single specific multi-processing module (MPM). Since we consider two different MPMs as *configuration options*, we need to compile *Apache httpd* twice for every *release*, once for the 'worker' MPM and once for the 'prefork' MPM. In addition to the respective MPM, we enabled the following modules during the compilation of *Apache httpd*: *ssl*, *deflate*, *socache-shmcb*.

Similar to the *HSQLDB* case study, we did not directly measure the *performance* as the execution time of *ab*. Instead we used the reported run time of *ab*.

PosgreSQL

To compile old *releases* of *PostgreSQL* with a new compiler, we had to disable aggressive loop optimizations with the compiler parameter `-fno-aggressive-loop-`

`optimizations`[17], to prevent segmentation faults at runtime. To prevent this missing optimization from having an unwanted influence on the *performance* of different *releases*, we disabled the optimization for all versions of *PostgreSQL*.

Since we used the same benchmark client as in the *HSQLDB* case study, we also used the same approach of extracting a value for the *performance* from the output of *PolePosition*.

libvpx VP8

To compile older *releases* of *libvpx VP8* with a modern compiler, we had to explicitly specify an older language standard of C89 with GNU extensions.

We measured the *performance* directly as the execution time of the encoder.

4.4 Operationalization

In this section, we formally describe the strategies we employ to evaluate our measurement results, which we will use in the following chapter to answer the research questions.

4.4.1 RQ1.1: Changes in Mean Performance and Energy

For our first research question, we investigate changes in the mean *performance* and mean *energy consumption* over all *configurations* for consecutive *releases*. We define the mean value of a *non-functional property* as follows:

$$\bar{x}_r = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} x_{r,c} \tag{4.1}$$

x is an *non-functional property*, either *performance* (p) or *energy consumption* (e). $\bar{x}_r$ is the mean value for a specific *release* $r \in \mathcal{R}$. $\mathcal{C}$ is the set of *configurations* and $x_{r,c}$ is the *non-functional property* value for a specific *release* r and *configuration* c.

We determine what we call *prominent releases*. We consider a *release* to be a *prominent release* for an *non-functional property* if there is a *substantial change* in that *non-functional property* from the preceding *release*.

A *substantial change* is a change in the value of a *non-functional property* that exceeds some threshold that is based on the relevant deviation. We define the absolute change $\Delta^{\bar{x}}_{r_i}$ in the mean value of a *non-functional property* x as the difference of mean values for two consecutive *releases* $r_{i-1}, r_i \in \mathcal{R}$, where r_{i-1} is the *release* directly preceding r_i:

$$\Delta^{\bar{x}}_{r_i} = \bar{x}_{r_i} - \bar{x}_{r_{i-1}} \tag{4.2}$$

For this research question, we consider one mean value for each *release* and *non-functional property*. Consequently, we also need one mean deviation value for each

[17] https://stackoverflow.com/a/34204352 – last visited on 2019-09-06

release and *non-functional property*. We use the root mean square (RMS) average over the deviation values of all relevant *configurations* to determine these mean deviation values.

$$\bar{\sigma}_r^x = \sqrt{\frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} (\sigma_{r,c}^x)^2} \tag{4.3}$$

$\bar{\sigma}_r^x$ is the mean deviation of the *non-functional property* x for a specific *release* r and $\sigma_{r,c}^x$ is the deviation of the *non-functional property* x for a specific *release* r and *configuration* c. The other symbols have the same meaning as in the formula above.

For a single *release*, we set the threshold to the mean standard deviation of that *release*. We then set the threshold $\theta_{r_i}^{\bar{x}}$ for *substantial changes* between two consecutive *releases* $r_{i-1}, r_i \in \mathcal{R}$, as the sum of the thresholds of these two *releases*, i.e. as the sum of mean deviations for these *releases*:

$$\theta_{r_i}^{\bar{x}} = \bar{\sigma}_{r_{i-1}}^x + \bar{\sigma}_{r_i}^x \tag{4.4}$$

With this approach, we can be certain that every *substantial change* we observe is in fact a change in the *non-functional property* value and not only caused by inaccurate measurements, as might be the case when the change does not exceed the deviations.

Formally, the set $\mathcal{R}_{\text{prom}}^x \subset \mathcal{R}$ of *prominent releases* for a *non-functional property* x is defined as follows, using the symbols defined above and the absolute value function *abs*:

$$\mathcal{R}_{\text{prom}}^x = \{\, r \in \mathcal{R} \mid abs(\Delta_r^{\bar{x}}) > \theta_r^{\bar{x}} \,\} \tag{4.5}$$

Once we have determined the *prominent releases* for both *performance* and *energy consumption*, we determine how many *prominent releases* there are for each *non-functional property* and if the *releases* are the same for both *non-functional properties*.

For *releases* that are *prominent releases* for both *performance* and *energy consumption*, we will compare the magnitude of the changes. Since we cannot directly compare the absolute values of these two *non-functional properties* with different units, we will determine relative change values:

$$\delta_{r_i}^{\bar{x}} = \frac{\bar{x}_{r_i} - \bar{x}_{r_{i-1}}}{\bar{x}_{r_{i-1}}} = \frac{\Delta_{r_i}^{\bar{x}}}{\bar{x}_{r_{i-1}}} \tag{4.6}$$

$\Delta_{r_i}^{\bar{x}}$ is the absolute change in the mean value for *non-functional property* x from the *release* $r_{i-1} \in \mathcal{R}$ preceding r_i to the *release* $r_i \in \mathcal{R}$ as defined above. Analogously, $\delta_{r_i}^{\bar{x}}$ is the relative change.

4.4.2 RQ1.2: Changes in Performance and Energy of Configurations

For the second research question, we increase the granularity and examine the *performance* and *energy consumption* of individual *configurations* rather than the mean over all *configurations*.

Similar to the previous research question, we find *configurations* that have a *substantial change* in *performance* or *energy consumption* between two consecutive *releases*. Again, the change in the value of a *non-functional property* x is defined simply as the difference between the values of the *non-functional property* for two consecutive *releases* $r_{i-1}, r_i \in \mathcal{R}$.

$$\Delta^x_{r_i,c} = x_{r_i,c} - x_{r_{i-1},c} \tag{4.7}$$

$x_{r,c}$ is the value of the *non-functional property* for a specific *release* r and *configuration* c.

Since we are now investigating individual *configurations*, we have to define a different threshold for which changes we consider to be *substantial*. We set the threshold for an individual *configuration* and *release* to the standard deviation of the measurements for that *configuration* and *release*. For the change between two consecutive *releases* $r_{i-1}, r_i \in \mathcal{R}$, we set the threshold $\theta^x_{r_i,c}$ for a *configuration* $c \in \mathcal{C}$ to the sum of the thresholds of the two *releases* for that *configuration*, i.e. the sum of the respective standard deviations:

$$\theta^x_{r_i,c} = \sigma^x_{r_{i-1},c} + \sigma^x_{r_i,c} \tag{4.8}$$

Since there are too many *configurations* to examine every single one of them, we only focus on the number of *configurations* with *substantial changes* for each *release*, only distinguishing between positive and negative change values. First, we define the sets $\mathcal{C}^{x,+}_{\text{prom},r}, \mathcal{C}^{x,-}_{\text{prom},r} \subset \mathcal{C}$ of *prominent configurations* for *non-functional property* x and the *releases* $r \in \mathcal{R}$ with respectively positive and negative changes that exceed the relevant threshold value:

$$\begin{aligned} \mathcal{C}^{x,+}_{\text{prom},r} &= \left\{ c \in \mathcal{C} \,\middle|\, \Delta^x_{r,c} > \theta^x_{r,c} \right\} \\ \mathcal{C}^{x,-}_{\text{prom},r} &= \left\{ c \in \mathcal{C} \,\middle|\, \Delta^x_{r,c} < -\theta^x_{r,c} \right\} \end{aligned} \tag{4.9}$$

Then we can count the *configurations* in each of these sets. We denote these numbers as N:

$$\begin{aligned} N^{x,+}_r &= \left|\mathcal{C}^{x,+}_{\text{prom},r}\right| \\ N^{x,-}_r &= \left|\mathcal{C}^{x,-}_{\text{prom},r}\right| \end{aligned} \tag{4.10}$$

Finally we can also determine the number of *configurations* that are *prominent* for a specific *release* for both *performance* (p) and *energy consumption* (e). We denote these numbers as N^{comm} (for ‘common’):

$$\begin{aligned} N_r^{\text{com},+} &= \left|\mathcal{C}_{\text{prom},r}^{p,+} \cap \mathcal{C}_{\text{prom},r}^{e,+}\right| \\ N_r^{\text{com},-} &= \left|\mathcal{C}_{\text{prom},r}^{p,-} \cap \mathcal{C}_{\text{prom},r}^{e,-}\right| \end{aligned} \tag{4.11}$$

4.4.3 RQ1.3: Changes in Performance and Energy of Features

In the third research question, we increase the amount of usable information by increasing the level of abstraction. Instead of considering individual *configurations*, we now investigate the influences of *configuration options* and combinations of *configuration options* on the *performance* and *energy consumption*.

To obtain that information from our measurements, we use *SPL Conqueror*. We proceed in five steps to obtain comparable *performance-influence and energy-influence models*.

In a first step, we use the iterative learning algorithm of *SPL Conqueror* to generate *performance-influence and energy-influence models* for each *release*. This results in models with different terms, in particular between *performance* and *energy consumption*, but also between different *releases*. We take the union of the terms from all models and proceed with the next step.

In the second step, we manually investigate the set of terms obtained in the first step to remove and replace terms according to the following rules:

- Remove terms that only appear in one or two *releases* and have a very small influence, because such terms are likely just noise and provide no useful information during later analysis of the generated models.
- When *configuration options* from an alternative group in the *feature model* appear in terms, remove terms with one option if all options from that alternative group appear, or add terms with options from the alternative group until all but one of the options is used in a term. Since options in an alternative group are all mutually exclusive, but one is always present, there are multiple equivalent representations of the same model. For example, for an alternative group with the options A and B, the two model formulas $10 + 5 \cdot A$ and $15 - 5 \cdot B$ represent the same model. By fixing, for all generated models, which options from an alternative group may appear, we ensure that all generated models use comparable representations.
- When terms make it clear that a learned influence is not dependent on the value of a numeric option but rather on the presence of a numeric option, replace occurrences of the numeric option with the parent option from the *feature model*. This ensures, similar to the previous rule, that out of multiple possible representations for the same model, comparable representations are used for all models.

Additionally, we manually investigate the measurement results and try to find influences from options and option interactions that are not already included in the set of terms from the first step, and add these to our set of model terms.

In the third step, we use the fitting algorithm of *SPL Conqueror* to generate *influence models* using the set of terms from the second step. This ensures that all models use the same terms regardless of *non-functional property* or *release*, but many terms will have a negligible influence in many of the models.

In the fourth step, we remove terms from each model that have only a very small influence. For this purpose, we calculate a threshold value for each model and remove terms with an absolute value smaller than the threshold. *SPL Conqueror* determines a *learning error* for the models it generates. We multiply this error rate with the mean value for the respective *non-functional property* and *release*, $\bar{x}_r$ (defined in Section 4.4.1), to obtain our threshold value.

In a final fifth step, we remove terms that we manually added in the second step but do not appear to be relevant in the fitted models from the fourth step. Then we repeat the fourth step to re-generate the models.

4.4.4 RQ2.1: Changes in the Correlation

In RQ2, we no longer investigate *performance* and *energy consumption* individually, but instead investigate the *correlation* between them. Since we are now directly comparing *performance* and *energy consumption* in that we are performing calculations involving both *non-functional properties* at the same time, we need to ensure that these properties are directly comparable. For this purpose, we will use the *fixed time energy consumption* for *infrastructure software* case studies.

In research question 2.1, we investigate only whole *releases*, i.e. we do not consider individual *configurations*, similar to our first research question.

We calculate the *Pearson correlation coefficient* ρ_r for each *release* $r \in \mathcal{R}$ from the *performance* and (fixed) *energy consumption* values of all *configurations* $c \in \mathcal{C}$:

$$\rho_r = \rho(p_{r,c}, e^f_{r,c}) \; , \; c \in \mathcal{C} \tag{4.12}$$

$abs(\rho)$	interpretation
≈ 1.0	perfect
> 0.8	very strong
> 0.6	strong
> 0.4	moderate
> 0.2	weak
≤ 0.2	none

Table 4.2: Interpretation of correlation coefficients

The numeric *Pearson correlation coefficient* is far more precise than the accuracy of the measurements. Generally, a *correlation* coefficient is usually interpreted with a rather low granularity. We use the interpretation shown in Table 4.2.

As seen in the results of related work, *correlation* between *performance* and *energy consumption* is often dominated by specific *configuration options*. In cases where we can identify such *configuration options*, we additionally investigate the *correlation* of only *configurations* without those options, allowing us to see potential changes in the *correlation* that are masked by the dominance of few *configuration options*. We specify a *constraint* s to select such a subset $\mathcal{C}_s \subset \mathcal{C}$ of the *configuration space*. We denote as $\rho_{r,s}$ the *Pearson correlation coefficient* between *performance* and *energy consumption* for a specific *release* $r \in \mathcal{R}$ of all selected configurations $c \in \mathcal{C}_s$.

$$\rho_{r,s} = \rho(p_{r,c}, e^{f}_{r,c}) \,,\ c \in \mathcal{C}_s \tag{4.13}$$

We investigate changes in the *correlation* between consecutive *releases*. A change $\Delta^{\rho}_{r_i}$ between two consecutive *releases* $r_{i-1}, r_i \in \mathcal{R}$ is the difference between the absolute values of the respective *Pearson correlation coefficients*. We take the absolute value of the *Pearson correlation coefficients*, since we are not interested in the direction but only the strength of the *correlation*.

$$\Delta^{\rho}_{r_i} = abs(\rho_{r_i}) - abs(\rho_{r_{i-1}}) \tag{4.14}$$

We are interested in *prominent releases*, i.e. *releases* with a *substantial change*. For the *correlation*, we consider a change to be *substantial* when the *correlation* changes by more than one level according to the interpretation (Table 4.2), i.e. a change by more than 0.2. The set $\mathcal{R}^{\rho}_{\text{prom}}$ with respect to the *Pearson correlation coefficient* is thus defined as follows:

$$\mathcal{R}^{\rho}_{\text{prom}} = \{\, r \in \mathcal{R} \mid abs(\Delta^{\rho}_{r}) > 0.2 \,\} \tag{4.15}$$

4.4.5 RQ2.2: Changes in the Correlation of Features

For the last research question, we increase the granularity from the previous research question and no longer investigate the *correlation* between *performance* and *energy consumption* only for all *configurations* of each *release* at once, but instead we investigate the *correlation* of different subsets of the *configuration space*, to investigate the *correlation* behaviour of specific *configuration options*. We consider one subset for each value of each *configuration option*, with the exception of alternative groups, where we consider one subset for each alternative. Formally, we define *constraints* to define these subsets and then use the same definition of the *correlation* $\rho_{r,s}$ in a subset $\mathcal{C}_s \subset \mathcal{C}$ of the *configuration space* as in the previous research question (Equation 4.13).

Due to the large number of *correlation* values we obtain with this approach, we will not compare all values. Also since *correlation* values have a rather low accuracy, we will again use the interpretation outlined in Table 4.2. As before, we consider a change in the *correlation* between two *releases* to be *substantial* if it exceeds 0.2.

5. Evaluation

In this chapter, we answer our research questions by evaluating the results of our case studies. In each section of this chapter, we focus on one of the research questions. First, we apply our operationalization as outlined in Section 4.4 and present the results. Then, we discuss the implications of the results and consider relations to the results of other research questions. Finally, we conclude each section with a summarized answer to the respective research question.

5.1 RQ1.1: Changes in Mean Performance and Energy

RQ1.1: Are there changes in the mean performance and mean energy consumption across releases?

Results

For the first research question, we examine the mean values of *performance* and *energy consumption* for each *release*.

In our first case study, *HSQLDB*, we discovered two *prominent releases* for *performance* and a single *prominent release* for *energy consumption*. We visualize the results for this case study in Figure 5.1. In the plot, every data point is the mean value of a *non-functional property* for a *release*. Different colours (styles) denote the two *non-functional properties*, *performance* in blue (dashed line) and *energy consumption* in red (dotted line). *Performance* values are given in seconds, *energy consumption* values in kilojoules. The lines between the data points indicate the changes in these values. The filled rectangles around each data point indicate the threshold values, i.e. the rectangles are vertically centred around the data points and have a height of twice the respective threshold values $\theta_r^{\bar{x}}$, once upwards for positive changes and once downwards for negative changes. Graphically, *substantial changes* are indicated by the lines between two data points leaving the respective boxes. On the other hand, if a line does not leave either box, a change is not *substantial* since

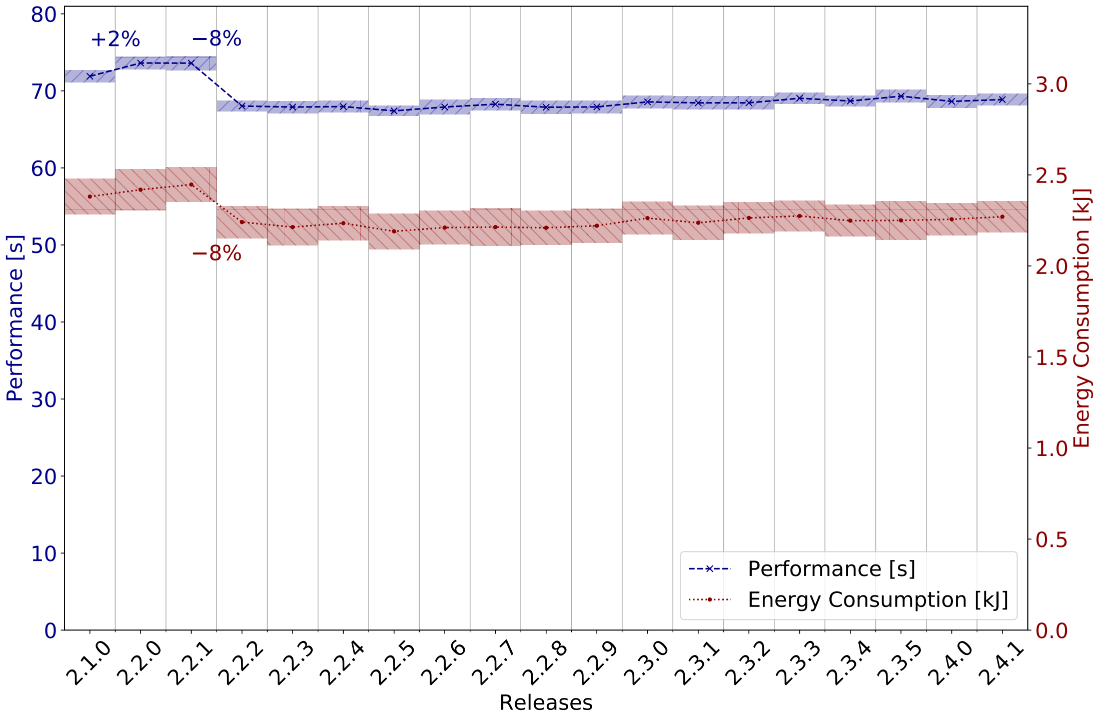


Figure 5.1: Mean *performance* and *energy* of the *HSQLDB* case study. Boxes indicate threshold values. Percentages indicate relative changes for *substantial changes*.

it does not exceed the sum of the two thresholds. Changes that are *substantial* are additionally labelled with the relative change $\delta_r^{\bar{x}}$ as percentage.

In the second case study, *Apache httpd*, we did not find any *prominent releases* for either *performance* or *energy consumption*. We do not show the corresponding result plot here since it does not provide any information relevant to this research question. Essentially, the plot shows two almost horizontal lines with changes never exceeding the thresholds. We include the plot in Section A.3 of the appendix for reference.

We do, however, show the results of our third case study, *PostgreSQL*, in Figure 5.2. For this case study, we found four and two *prominent releases* for *performance* and *energy consumption*, respectively. The plot is structured in the same way as the one for the *HSQLDB* case study.

We found the largest number of *prominent releases* in our final case study, *libvpx VP8*, with eleven *substantial changes* in the *performance* and four in the *energy consumption*. We show the results in a plot following the same schema as the previous ones in Figure 5.3.

To summarize the results for this research question, we list all *prominent releases* for both *performance* p and *energy consumption* e of each case study in Table 5.1.

Discussion

In the *HSQLDB* case study, we can see a small positive *substantial change* with *release* 2.2.0, which is only visible in the *performance*. In *release* 2.2.2, we can see

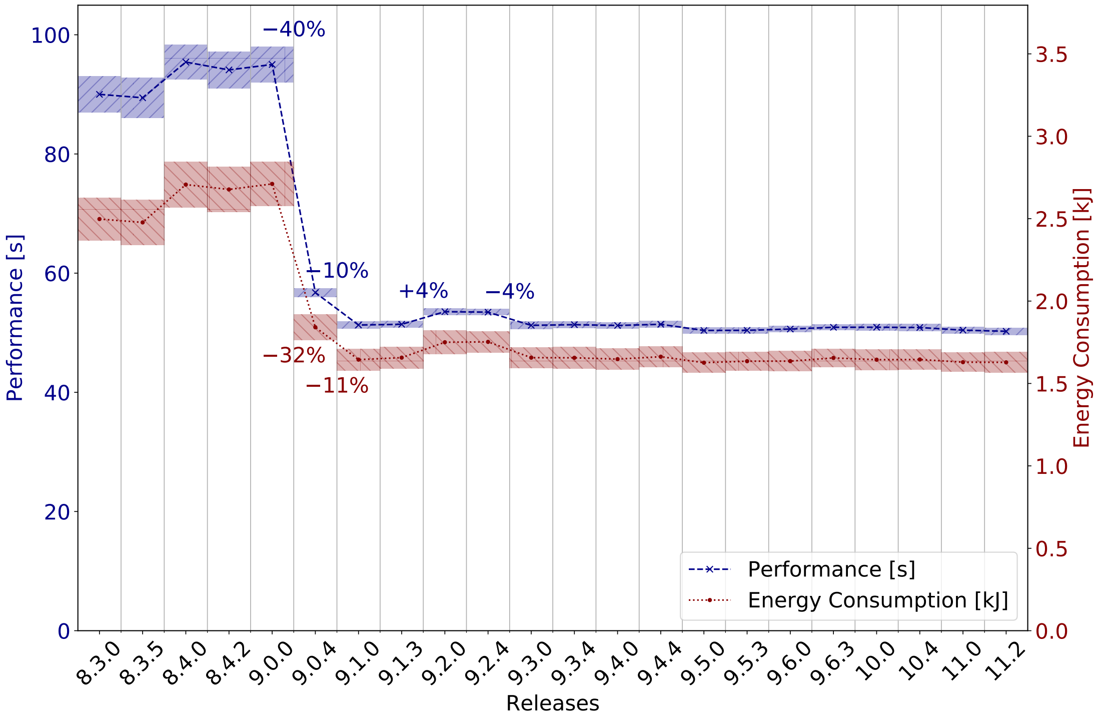


Figure 5.2: Mean *performance* and *energy* of the *PostgreSQL* case study. Boxes indicate threshold values. Percentages indicate relative changes for *substantial changes*.

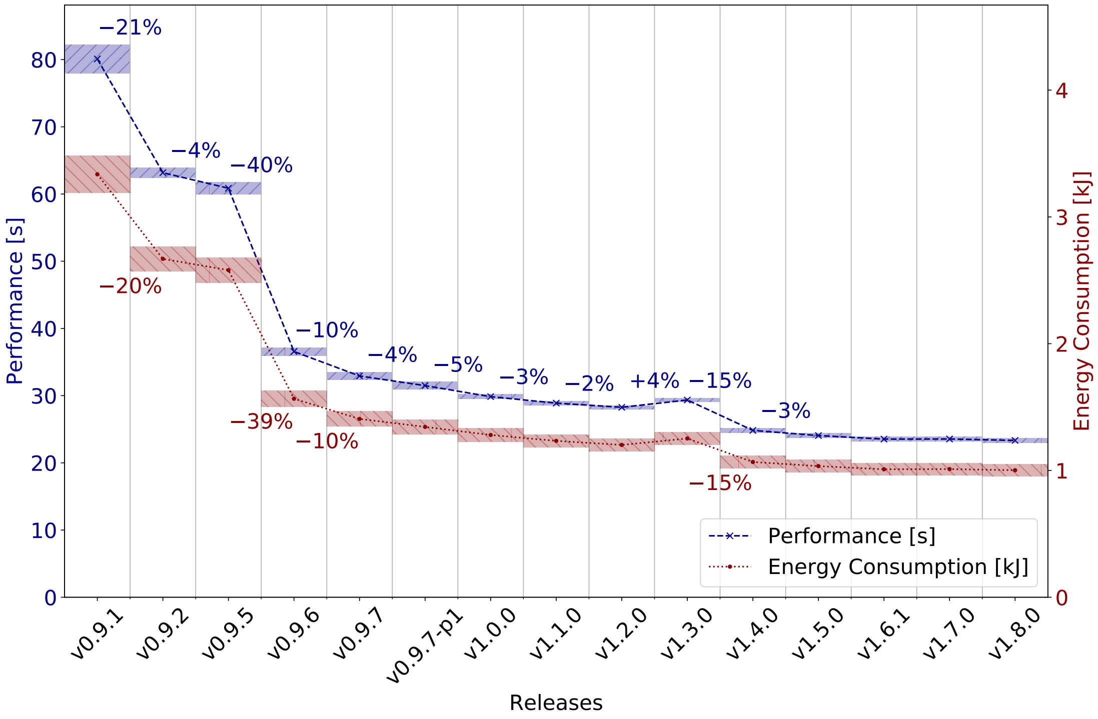


Figure 5.3: Mean *performance* and *energy* of the *libvpx VP8* case study. Boxes indicate threshold values. Percentages indicate relative changes for *substantial changes*.

Case study	$\mathcal{R}^p_{\text{prom}}$	$\mathcal{R}^e_{\text{prom}}$
HSQLDB	2.2.0, 2.2.2	2.2.2
Apache httpd	$\emptyset$	$\emptyset$
PostgreSQL	9.0.4, 9.1.0, 9.2.0, 9.3.0	9.0.4, 9.1.0
libvpx VP8	0.9.2, 0.9.5, 0.9.6, 0.9.7, 0.9.7-p1, 1.0.0, 1.1.0, 1.2.0, 1.3.0, 1.4.0, 1.5.0	0.9.2, 0.9.6, 0.9.7, 1.4.0

Table 5.1: *Prominent releases* for both *performance* and *energy consumption* of each case study

a much larger negative change that presents itself with an equal relative value in both *non-functional properties*. All other *releases* do not show any *substantial* or otherwise noticeable changes in the mean of all *configurations*.

The *Apache httpd* case study shows us, that there is software that evolves but for which changes do not necessarily impact the *performance* or *energy consumption* at all. Due to our relatively simple workload in this case study, it is likely that changes in the different *releases* of *Apache httpd* generally had an impact on *performance* and *energy consumption*, but our workload simply did not utilize the functionality affected by these changes.

The third case study, *PostgreSQL*, paints a similar picture to the *HSQLDB* case study, with several smaller changes that are only substantial in the *performance* and one large negative change. In *release* 8.4.2, we can see a noticeable positive change in both *performance* and *energy consumption*, that is however just below the thresholds and thus not *substantial* according to our definition. There is a very large negative change in *release* 9.0.4, with considerably differing values of −40% for *performance* and only −32% for *energy consumption*. In the next *release*, 9.1.0, follows a smaller but still relatively large negative change that is almost the same for *performance* and *energy consumption*. After that, we see two smaller *performance*-only *substantial changes*, one positive and one negative, in *releases* 9.2.0 an 9.2.3, respectively.

The *application software* case study, *libvpx VP8*, has by far the most *prominent releases*. Nevertheless, we can see similarities to the previous case studies. Changes that are very small, are only *substantial* in the *performance*, but may still be noticeable in the *energy consumption*. Larger changes are represented with similar relative change values in both *non-functional properties*. Out of the many changes we can see for this case study, only a single one with a small value is positive and all others are negative.

As shown in Table 5.1, every *release* that is *prominent* for *energy consumption* is also a *prominent release* for *performance*. In almost all of these cases, the relative changes (values shown in the plots) are equal or almost equal between *performance* and *energy consumption*.

There is only one notable exception with the *release* 9.0.4 of *PostgreSQL*. The relative change in the *performance* is −40%, while the relative change in the *energy* *consumption* is only −32%. We have identified the *configuration option* 'fsync' as the primary reason for the change in this *release*. This *feature* essentially requests *PostgreSQL* to wait for file changes to be physically written to disk before completing an operation. In other words, 'fsync' just tells *PostgreSQL* to wait for the disk. While waiting for an amount of time has a direct influence on the *performance*, which is just time, the system is effectively idling while waiting and consequently the influence on the *energy consumption* is comparably low.

We can see, that whenever a *release* has a *substantial change* for *performance* but not for *energy consumption*, the relative change in the *performance* is relatively small (at most 5%). In most of these cases, the plots show a clear tendency in the *energy consumption* that resembles the change in the *performance*, but is still within the threshold range. We assume that in these situations, changes in the *energy consumption* also resemble changes in the *performance*, but we cannot be certain because of the higher deviations in the energy measurements.

Over all case studies, we investigated a combined total of 73 pairs of consecutive *releases*. Out of those, we found 17 *releases* (23% of investigated *releases*) with *substantial changes* in the *performance* and 7 *releases* (10%) with *substantial changes* in the *energy consumption*. Since all *releases* with *substantial changes* in the *energy consumption* also have *substantial changes* in the *performance*, this amounts to 100% of all *prominent releases* being *prominent* for the *performance* and 41% for the *energy consumption*.

Another observation that applies equally to all of our case studies, is that positive changes in *performance* and *energy consumption* tend to be small while all magnitudes of change values appear for negative changes. This makes sense from a software developer's point of view, since improvements in *performance* are appreciated and specific improvements may even be implemented on purpose, whereas a *performance* regression will only be tolerated if absolutely necessary, e.g. for security reasons.

The largest positive changes we observed, are two changes by 4% in the *performance* in version 9.2.0 of *PostgreSQL* and v1.3.0 of *libvpx VP8*. We did not observe any *substantial* positive changes in the *energy consumption*. The largest negative changes we observed, are two changes by −40% in the *performance* in version 9.0.4 of *PostgreSQL* and v0.9.6 of *libvpx VP8* and a change by −39% in the *energy consumption* of v0.9.6 of *libvpx VP8*.

> We found changes in the *performance* and *energy consumption* in specific *releases* for three of the four software systems we investigated. 23% of measured *releases* had a *substantial change* in the *performance* and 10% in the *energy consumption*. We observed positive changes of up to 4% in the *performance* and negative changes of up to −40% in the *performance* and −39% in the *energy consumption*. Out of all measured *releases* with *substantial changes*, 100% had *substantial changes* in the *performance* and 41% had *substantial changes* in the *energy consumption*.

5.2 RQ1.2: Changes in Performance and Energy of Configurations

RQ1.2: Are there changes in the performance and energy consumption of individual configurations across releases?

Results

For the second research question, we investigate the *performance* and *energy consumption* of individual *configurations* and in particular their differences between consecutive *releases*.

We visualize the results for the *HSQLDB* case study in Figure 5.4. The figure consists of two plots, one for *performance* and one for *energy consumption*. The plots are grids with columns for the *configurations* and rows for the *releases*. The values associated with each cell of the grid are shown with different colours. Absolute values ($x_{r,c}$) are shown for the first *release*, for all other *releases*, the difference ($\Delta^x_{r,c}$) to the previous *release* is shown. Only differences that exceed the respective threshold ($\theta^x_{r,c}$) are coloured, all other cells are blank. The *configurations* in both plots are ordered by the *performance* of the first *release*.

For this case study we observe *substantial changes* in every *release*. However, only a small number of *configurations* and *releases* has large changes, with most other changes barely exceeding the thresholds.

Due to the large number of *configurations* we do not examine every single change value but instead count the *prominent configurations* for each *release*, only distinguishing between *performance* and *energy consumption*, as well as between positive and negative changes.

We show these numbers for the *HSQLDB* case study in Table 5.2. There are many more *substantial changes* in the *performance* than in the *energy consumption*. Almost all *configurations* that have *substantial changes* in the *energy consumption*, also have *substantial changes* in the *performance*, as can be seen in the table from comparing the numbers for *energy consumption* (e) and the common numbers (comm).

In the same way, we show the results for the *Apache httpd* case study in Figure 5.5. For this case study, we can see considerable differences in the plots for *performance* and *energy consumption*. There are again *substantial changes* in all *releases*. None of the changes are particularly large compared to the other case studies, with most of them being just above the thresholds.

Table 5.3 shows the numbers of *prominent configurations* for each *release*. Again, we can see that there are many more *substantial changes* in the *performance* than in the *energy consumption*. In this case study, there are many *configurations* with a *substantial change* in the *energy consumption* that do not at the same time have a *substantial change* in the *performance*.

We present the results for the third case study, *PostgreSQL*, in Figure 5.6. We can see that the plots for *performance* and *energy consumption* look almost the same,

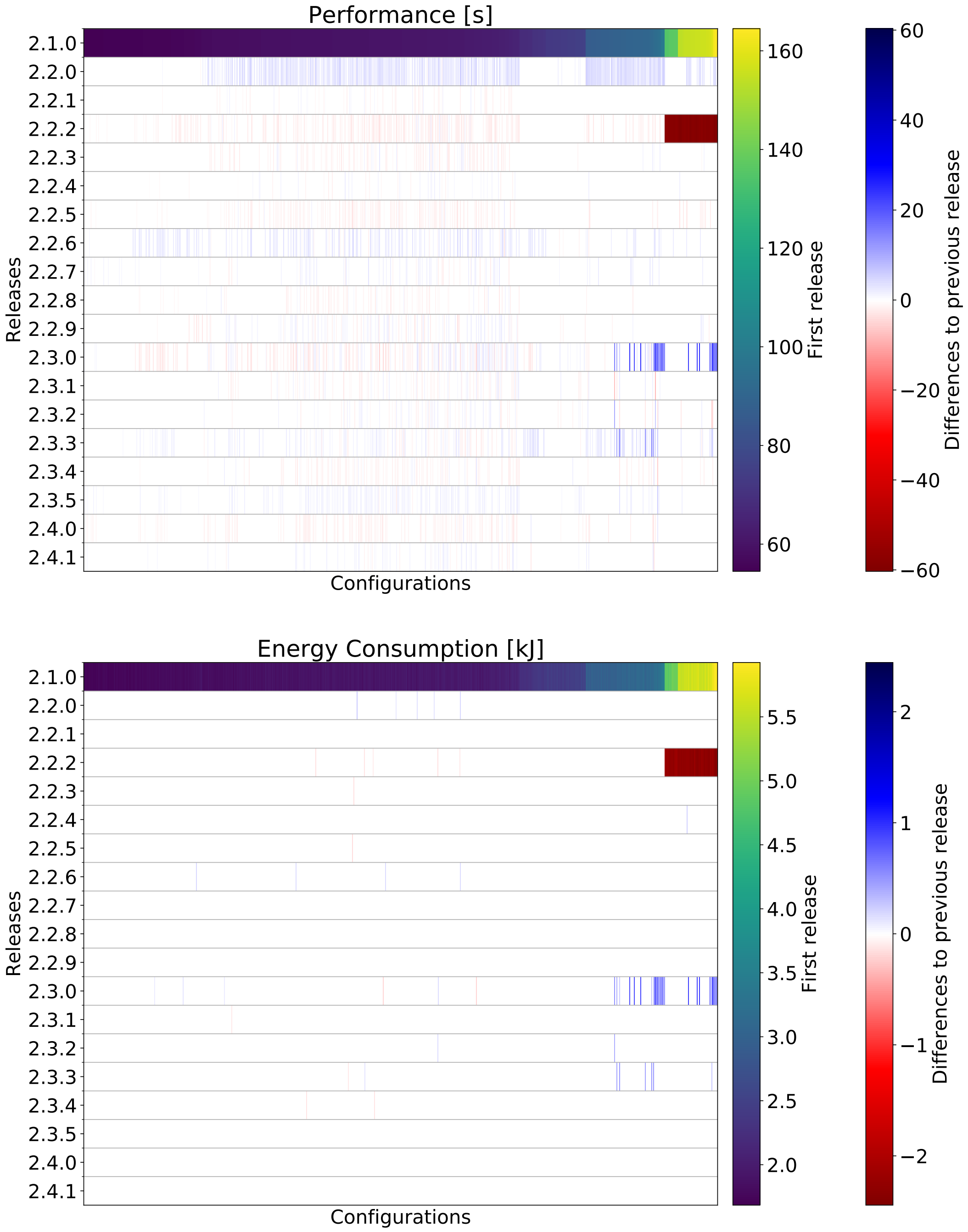


Figure 5.4: *Performance* and *energy consumption* for *configurations* and *releases* of the *HSQLDB* case study. Absolute values are shown for the first *release* and differences to the previous *release* for all other *releases*. Configurations are ordered by the *performance* of the first *release* for all *releases* and for both the *performance* and *energy consumption* plots. Only values exceeding the threshold are shown.

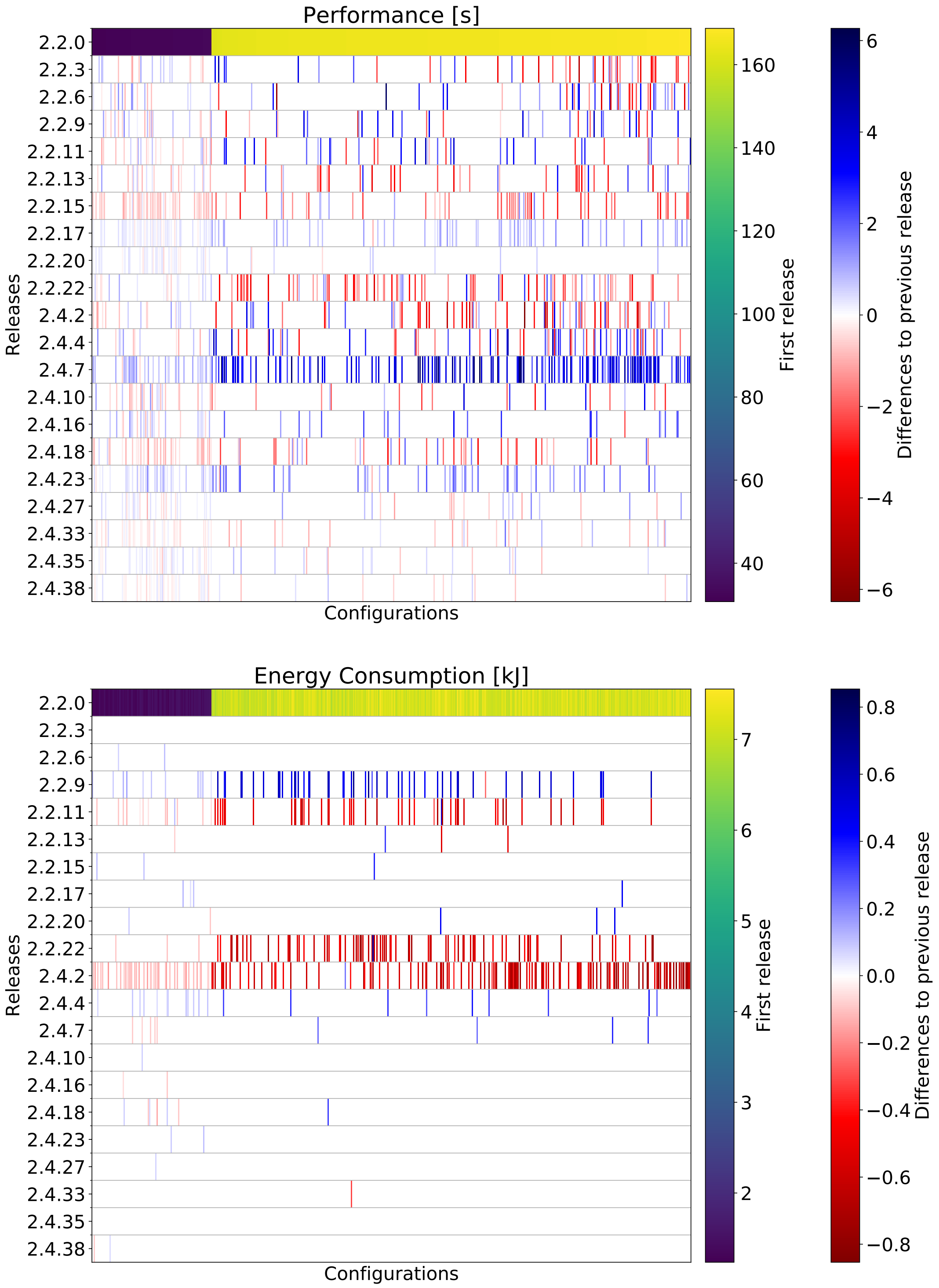


Figure 5.5: *Performance* and *energy consumption* for *configurations* and *releases* of the *Apache httpd* case study. Absolute values are shown for the first *release* and differences to the previous *release* for all other *releases*. Configurations are ordered by the *performance* of the first *release* for all *releases* and for both the *performance* and *energy consumption* plots. Only values exceeding the threshold are shown.

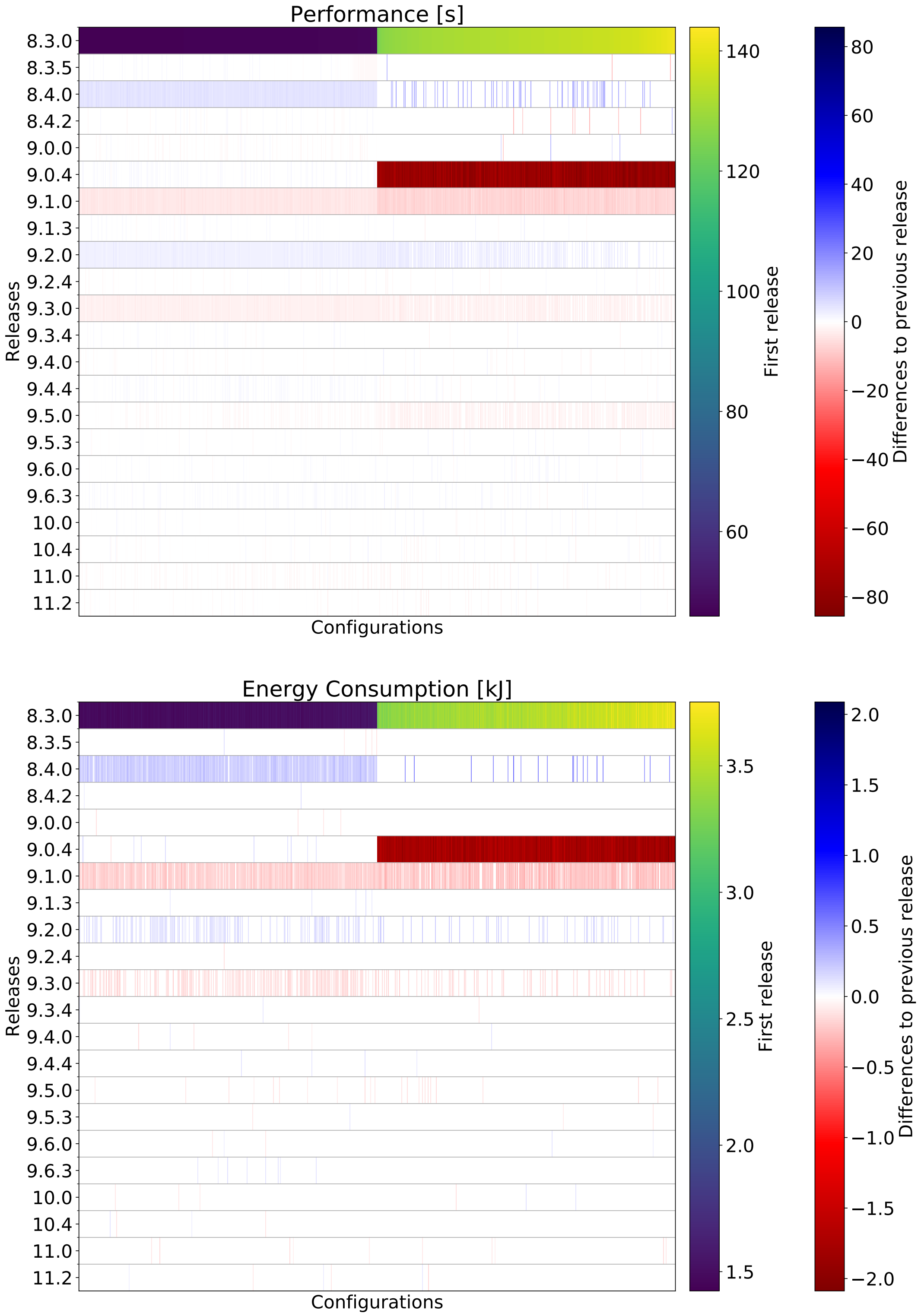


Figure 5.6: *Performance* and *energy consumption* for *configurations* and *releases* of the *PostgreSQL* case study. Absolute values are shown for the first *release* and differences to the previous *release* for all other *releases*. Configurations are ordered by the *performance* of the first *release* for all *releases* and for both the *performance* and *energy consumption* plots. Only values exceeding the threshold are shown.

Release r	$N_r^{p,+}$	$N_r^{p,-}$	$N_r^{e,+}$	$N_r^{e,-}$	$N_r^{\text{comm},+}$	$N_r^{\text{comm},+}$
2.1.0	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
2.2.0	447 (51%)	9 (1%)	5 (0%)	0 (0%)	5 (0%)	0 (0%)
2.2.1	18 (2%)	44 (5%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
2.2.2	14 (1%)	297 (34%)	0 (0%)	77 (8%)	0 (0%)	77 (8%)
2.2.3	27 (3%)	100 (11%)	0 (0%)	1 (0%)	0 (0%)	1 (0%)
2.2.4	28 (3%)	18 (2%)	1 (0%)	0 (0%)	0 (0%)	0 (0%)
2.2.5	10 (1%)	178 (20%)	0 (0%)	1 (0%)	0 (0%)	1 (0%)
2.2.6	175 (20%)	17 (1%)	4 (0%)	0 (0%)	4 (0%)	0 (0%)
2.2.7	65 (7%)	23 (2%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
2.2.8	12 (1%)	74 (8%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
2.2.9	78 (9%)	49 (5%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
2.3.0	143 (16%)	132 (15%)	38 (4%)	2 (0%)	35 (4%)	2 (0%)
2.3.1	39 (4%)	75 (8%)	0 (0%)	1 (0%)	0 (0%)	1 (0%)
2.3.2	48 (5%)	42 (4%)	2 (0%)	0 (0%)	2 (0%)	0 (0%)
2.3.3	161 (18%)	45 (5%)	7 (0%)	1 (0%)	7 (0%)	1 (0%)
2.3.4	16 (1%)	109 (12%)	0 (0%)	2 (0%)	0 (0%)	1 (0%)
2.3.5	176 (20%)	3 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
2.4.0	6 (0%)	155 (17%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
2.4.1	65 (7%)	17 (1%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)

Table 5.2: Number of prominent *configurations* for each *release* of the *HSQLDB* case study, individually for positive (+) and negative (−) changes, as well as for *performance* (p), *energy consumption* (e) and commonly (comm).

even though they describe two different *non-functional properties*. Only few *releases* show noticeable changes in a large number of *configurations*.

We show the numbers of *prominent configurations* for *PostgreSQL* in Table 5.4. We can see that overall there are many more changes in the *performance* than in the *energy consumption*, although for some *releases* there are almost as many changes in the *energy consumption* as in the *performance*. Again, similarly to the *HSQLDB* case study, most *configurations* with *substantial changes* in the *energy consumption* also have *substantial changes* in the *performance*.

For the final case study, *libvpx VP8*, we show the plots in Figure 5.7. Just like in the previous case study, the plots for *performance* and *energy consumption* look almost the same. Almost all changes are negative. There are few *releases* with hardly any changes but most *releases* show *substantial changes* in many *configurations*.

The corresponding numbers of *prominent configurations* are shown in Table 5.5. We can see a much larger number of *configurations* with *substantial changes* than in the other case studies, which is only partially caused by this case study having a larger *configuration space* than the previous case studies. In some of the *releases*, more than half of all 2 736 *configurations* have *substantial changes*. The overlap between *prominent configurations* for *performance* and *energy consumption* is large for some *releases* but small for others.

Release r	$N_r^{p,+}$	$N_r^{p,-}$	$N_r^{e,+}$	$N_r^{e,-}$	$N_r^{\text{comm},+}$	$N_r^{\text{comm},+}$
2.2.0	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
2.2.3	24 (3%)	34 (5%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
2.2.6	33 (5%)	23 (3%)	2 (0%)	0 (0%)	1 (0%)	0 (0%)
2.2.9	32 (5%)	22 (3%)	60 (9%)	1 (0%)	6 (0%)	0 (0%)
2.2.11	34 (5%)	24 (3%)	2 (0%)	58 (9%)	0 (0%)	2 (0%)
2.2.13	19 (2%)	38 (5%)	1 (0%)	3 (0%)	0 (0%)	0 (0%)
2.2.15	12 (1%)	91 (14%)	3 (0%)	0 (0%)	0 (0%)	0 (0%)
2.2.17	96 (15%)	9 (1%)	4 (0%)	0 (0%)	0 (0%)	0 (0%)
2.2.20	39 (6%)	12 (1%)	4 (0%)	1 (0%)	1 (0%)	0 (0%)
2.2.22	26 (4%)	73 (11%)	1 (0%)	80 (12%)	0 (0%)	30 (4%)
2.4.2	28 (4%)	54 (8%)	1 (0%)	166 (25%)	1 (0%)	16 (2%)
2.4.4	47 (7%)	33 (5%)	21 (3%)	0 (0%)	4 (0%)	0 (0%)
2.4.7	172 (26%)	2 (0%)	4 (0%)	5 (0%)	2 (0%)	0 (0%)
2.4.10	16 (2%)	29 (4%)	1 (0%)	0 (0%)	1 (0%)	0 (0%)
2.4.16	40 (6%)	9 (1%)	0 (0%)	2 (0%)	0 (0%)	1 (0%)
2.4.18	13 (2%)	65 (10%)	4 (0%)	3 (0%)	0 (0%)	2 (0%)
2.4.23	82 (12%)	5 (0%)	2 (0%)	0 (0%)	2 (0%)	0 (0%)
2.4.27	30 (4%)	18 (2%)	1 (0%)	0 (0%)	1 (0%)	0 (0%)
2.4.33	15 (2%)	46 (7%)	0 (0%)	1 (0%)	0 (0%)	0 (0%)
2.4.35	40 (6%)	17 (2%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
2.4.38	22 (3%)	29 (4%)	1 (0%)	1 (0%)	0 (0%)	1 (0%)

Table 5.3: Number of prominent *configurations* for each *release* of the *Apache httpd* case study, individually for positive (+) and negative (−) changes, as well as for *performance* (p), *energy consumption* (e) and commonly (comm).

Discussion

In the previous research question, we have seen for the *HSQLDB* case study a small positive change in the *performance* for version 2.2.0 and a larger negative change in both *performance* and *energy consumption* in version 2.2.2. These changes are reflected in the results from this research question. We can see that the change in the mean value of 2.2.0 results from relatively small changes in a large number of *configurations*. These changes are not visible in the *energy consumption*. Since the changes are small and barely above the threshold, it is possible that the changes exist in the energy consumption, but since deviations are higher for the *energy consumption* measurements, the changes are simply not exceeding the thresholds for the *energy consumption*. The change in 2.2.2, on the other hand, is caused by only a small number of *configurations* with large changes. We can see that this change is limited to *configurations* that are much slower than all other *configurations* in *releases* before 2.2.2. We can also see that the change value of approximately $-60s$ for the *performance* and $-2kJ$ for the *energy consumption* is similar to the difference in the absolute values between these and other *configurations*. This results in a much smaller range of values for both *non-functional properties* for all *configurations* in and after *release* 2.2.2.

Release r	$N_r^{p,+}$	$N_r^{p,-}$	$N_r^{e,+}$	$N_r^{e,-}$	$N_r^{\text{comm},+}$	$N_r^{\text{comm},+}$
8.3.0	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
8.3.5	30 (3%)	44 (5%)	1 (0%)	4 (0%)	1 (0%)	3 (0%)
8.4.0	490 (56%)	0 (0%)	434 (50%)	0 (0%)	431 (49%)	0 (0%)
8.4.2	26 (3%)	25 (2%)	2 (0%)	0 (0%)	1 (0%)	0 (0%)
9.0.0	12 (1%)	76 (8%)	0 (0%)	4 (0%)	0 (0%)	3 (0%)
9.0.4	112 (12%)	437 (50%)	10 (1%)	433 (50%)	10 (1%)	433 (50%)
9.1.0	0 (0%)	864 (100%)	0 (0%)	767 (88%)	0 (0%)	767 (88%)
9.1.3	58 (6%)	8 (0%)	5 (0%)	0 (0%)	2 (0%)	0 (0%)
9.2.0	677 (78%)	0 (0%)	185 (21%)	0 (0%)	177 (20%)	0 (0%)
9.2.4	16 (1%)	43 (4%)	0 (0%)	1 (0%)	0 (0%)	1 (0%)
9.3.0	0 (0%)	772 (89%)	0 (0%)	265 (30%)	0 (0%)	262 (30%)
9.3.4	36 (4%)	15 (1%)	2 (0%)	1 (0%)	1 (0%)	0 (0%)
9.4.0	16 (1%)	52 (6%)	2 (0%)	4 (0%)	0 (0%)	2 (0%)
9.4.4	129 (14%)	18 (2%)	4 (0%)	0 (0%)	3 (0%)	0 (0%)
9.5.0	1 (0%)	417 (48%)	0 (0%)	22 (2%)	0 (0%)	17 (1%)
9.5.3	24 (2%)	18 (2%)	1 (0%)	3 (0%)	0 (0%)	2 (0%)
9.6.0	65 (7%)	9 (1%)	3 (0%)	2 (0%)	0 (0%)	1 (0%)
9.6.3	133 (15%)	10 (1%)	8 (0%)	0 (0%)	6 (0%)	0 (0%)
10.0	38 (4%)	23 (2%)	2 (0%)	4 (0%)	2 (0%)	2 (0%)
10.4	25 (2%)	45 (5%)	2 (0%)	2 (0%)	1 (0%)	2 (0%)
11.0	5 (0%)	149 (17%)	0 (0%)	10 (1%)	0 (0%)	5 (0%)
11.2	11 (1%)	53 (6%)	3 (0%)	3 (0%)	0 (0%)	2 (0%)

Table 5.4: Number of prominent *configurations* for each *release* of the *PostgreSQL* case study, individually for positive (+) and negative (−) changes, as well as for *performance* (p), *energy consumption* (e) and commonly (comm).

In addition to the two *releases* we have found to be *prominent* in the previous research question, i.e. in the mean values, we can see that the *releases* 2.3.0 and 2.3.3 have *configurations* with relatively large changes. However, these do not sum up to large enough amounts to be visible in the mean values from the previous research question.

For the second case study, *Apache httpd*, we did not see any *substantial changes* in the mean values. Looking at individual *configurations*, we can see that many *configurations* show changes in some of the *releases*. However, there are many cases where there are both positive and negative changes in the same *release*, likely cancelling each other out in the mean value, and even in *releases* where there is a majority of changes that are only positive or negative, e.g. in 2.4.7 with mostly positive changes for the *performance* or 2.4.2 with mostly negative changes for the *energy consumption*, the absolute values of the changes are so small, that in the mean over all *configurations*, the relevant thresholds are not reached.

For the *PostgreSQL* case study, we have seen in RQ1.1, a positive change in both *non-functional properties* that is noticeable but not *substantial* in version 8.4.0. We observe that this change is *substantial* in approximately half of the *configurations* in this *release*. In 9.0.4, we identified a large negative change in the previous research

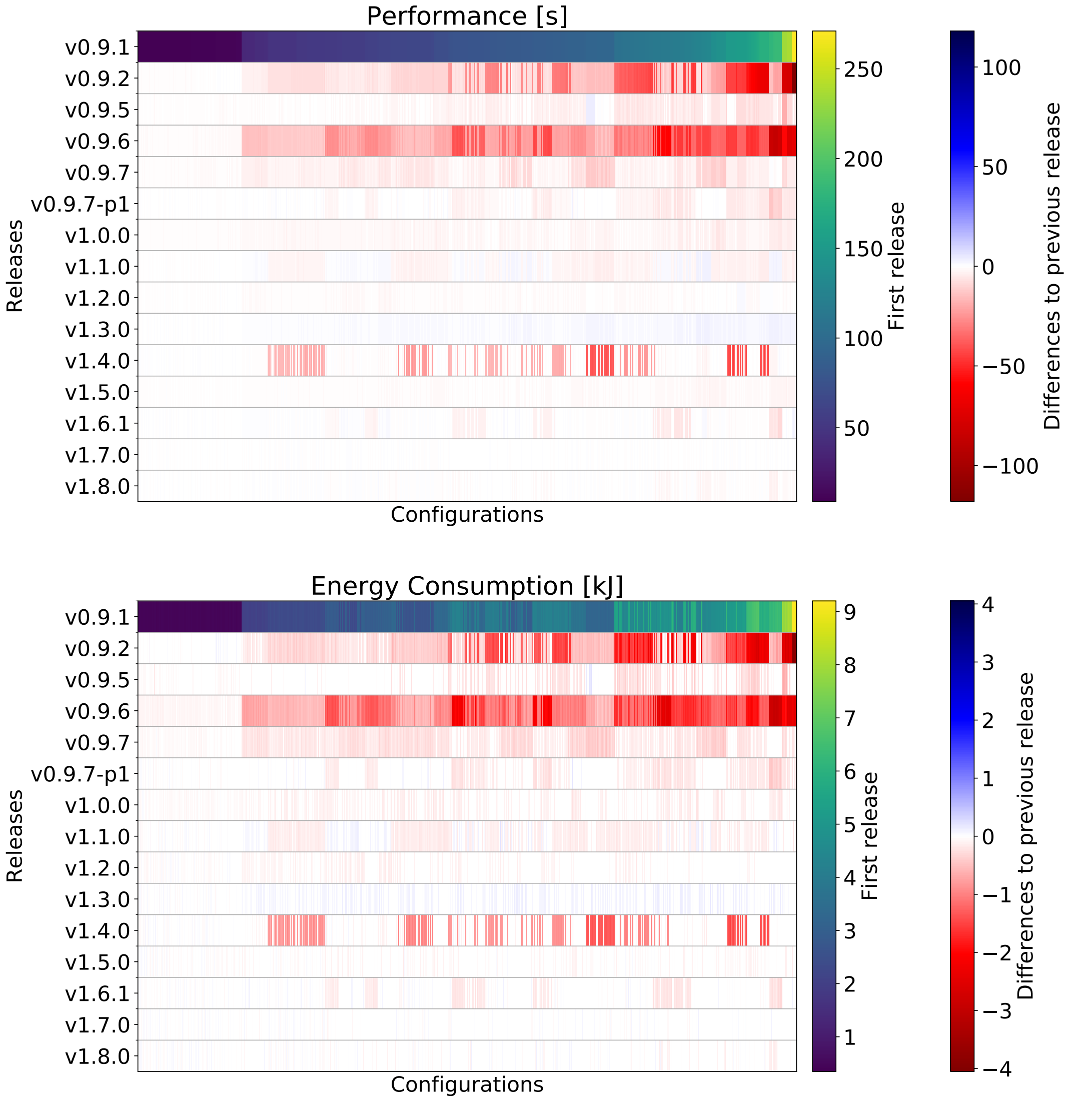


Figure 5.7: *Performance* and *energy consumption* for *configurations* and *releases* of the *libvpx VP8* case study. Absolute values are shown for the first *release* and differences to the previous *release* for all other *releases*. Configurations are ordered by the *performance* of the first *release* for all *releases* and for both the *performance* and *energy consumption* plots. Only values exceeding the threshold are shown.

question, and this change is also very apparent in the individual *configurations*, with exactly half of *configurations* showing a large negative change in this *release*. Those are the *configurations* that are slower than other *configurations* in *releases* before 9.0.4. Similar to the situation in *release* 2.2.2 of *HSQLDB*, all *configuration* in and after *release* 9.0.4 have *performance* and *energy consumption* values in a very small range. As already mentioned in the previous research question, the *configurations* with the large change in 9.0.4 are those with the option ‘fsync’, which explains why exactly half of the *configurations* are affected. The next change that is visible in the mean values, is a negative change in 9.1.0, which is *substantial* in almost all of the *configurations*. The remaining *prominent releases* have *substantial changes*

Release r	$N_r^{p,+}$	$N_r^{p,-}$	$N_r^{e,+}$	$N_r^{e,-}$	$N_r^{\text{comm},+}$	$N_r^{\text{comm},+}$
v0.9.1	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
v0.9.2	32 (1%)	2 553 (93%)	19 (0%)	1 964 (71%)	9 (0%)	1 962 (71%)
v0.9.5	48 (1%)	1 877 (68%)	11 (0%)	541 (19%)	9 (0%)	528 (19%)
v0.9.6	1 (0%)	2 729 (99%)	1 (0%)	2 631 (96%)	1 (0%)	2 629 (96%)
v0.9.7	10 (0%)	2 610 (95%)	5 (0%)	1 820 (66%)	0 (0%)	1 817 (66%)
v0.9.7-p1	190 (6%)	1 234 (45%)	43 (1%)	812 (29%)	25 (0%)	779 (28%)
v1.0.0	0 (0%)	2 499 (91%)	9 (0%)	709 (25%)	0 (0%)	703 (25%)
v1.1.0	862 (31%)	1 534 (56%)	216 (7%)	1 83 (39%)	179 (6%)	1 54 (38%)
v1.2.0	100 (3%)	1 987 (72%)	30 (1%)	456 (16%)	2 (0%)	407 (14%)
v1.3.0	2 358 (86%)	30 (1%)	539 (19%)	29 (1%)	497 (18%)	5 (0%)
v1.4.0	218 (7%)	1 117 (40%)	52 (1%)	588 (21%)	23 (0%)	542 (19%)
v1.5.0	14 (0%)	2 198 (80%)	70 (2%)	415 (15%)	4 (0%)	366 (13%)
v1.6.1	660 (24%)	876 (32%)	164 (5%)	511 (18%)	92 (3%)	408 (14%)
v1.7.0	288 (10%)	383 (13%)	148 (5%)	143 (5%)	43 (1%)	67 (2%)
v1.8.0	460 (16%)	848 (30%)	143 (5%)	212 (7%)	50 (1%)	109 (3%)

Table 5.5: Number of prominent *configurations* for each *release* of the *libvpx VP8* case study, individually for positive (+) and negative (−) changes, as well as for *performance* (p), *energy consumption* (e) and commonly (comm).

only in the *performance*, with a small positive change in 9.2.0 and a small negative change in 9.3.0. These changes are *substantial* in almost all *configurations* for the *performance*, and in some *configurations* also for the *energy consumption*, supporting our assumption that these changes are appearing in both *performance* and *energy consumption* alike, with changes in the *energy consumption* simply not – or not always – exceeding the threshold values due to the higher deviations of the energy measurements.

In the final case study, *libvpx VP8*, we can again see similarities between the changes in the mean values from the previous research question, and the changes in individual *configurations*. We have observed large changes in the mean values of v0.9.2 and v0.9.6. These changes are also very visible in the results from the individual *configurations*, with almost all *configurations* showing negative changes in both *non-functional properties* in these two *releases*. However, the changes have different magnitudes in the different *configurations*. Generally, we can observe all the changes that are visible in the mean values also in the individual *configurations*. Just like in the previous case study, we can see that these changes equally affect both *performance* and *energy consumption*, even in cases where the mean values did not clearly show these changes in the *energy consumption*. Additionally, we can observe there are almost no visible changes in both *performance* and *energy consumption* for approximately the first sixth of *configurations*. This is likely caused by the fact that these *configurations* have very small absolute values in both *performance* and *energy consumption*, resulting in a comparably high threshold, since the threshold is based on the deviation which tends to be relatively large for short measurements.

In addition to the plots, we can also examine the numbers of *prominent configurations* as shown in the tables. For *HSQLDB* (Table 5.2), comparing the *configurations* for *energy consumption* and common *configurations* for both *performance* and *energy consumption*, we can see that the numbers are equal or almost equal for all of the *releases*. This shows us, that almost all *configurations* that are *prominent* for the *energy consumption* are also *prominent* for the *performance*, but there are many *configurations* that are *prominent* for the *performance* but not for the *energy consumption*. This is in line with our previous observation of changes being generally the same for *performance* and *energy consumption*, but that changes in the *energy consumption* often do not exceed the thresholds due to higher deviations. For the few cases where *configurations* have *substantial changes* in the *energy consumption* but not in the *performance*, changes in the *energy consumption* are close to the thresholds, indicating that the respective *performance* changes may also exist but fall below the thresholds.

For *Apache httpd*, the numbers of *prominent configurations* shown in Table 5.3 paint a similar picture to the one from the previous case study, with the exception that a much larger number of *configurations* shows *substantial changes* in the *energy consumption* but not in the *performance*. This is not surprising since we already observed that all changes are small in this case study, so it is expected that many changes are not exceeding the thresholds for both *non-functional properties*.

For the third case study, *PostgreSQL*, the numbers shown in Table 5.4 closely resemble the results from the *HSQLDB* case study. Almost all of the *prominent configurations* for *energy consumption* are also *prominent* for the performance, with the few exceptions having very low change values.

In the final case study, *libvpx VP8*, the numbers of *prominent configurations* (Table 5.5) are not as straightforward to interpret as those of the other case studies, because there is a large number of *configurations* with *substantial changes* in every *release*. We can still see that for many *releases*, almost all of the *configurations* with *energy consumption* changes also have *performance* changes. While we have a much larger number of exceptions for this case study, i.e. *configurations* that have a *substantial change* in the *energy consumption* but not in the *performance*, almost all of these energy changes are very low, just like in the previous case studies.

Overall we can see – with the exception of the *Apache httpd* case study, where we have very small change values and consequently a comparably large influence from noise – similar observations in all case studies. Changes in the mean values are reflected in the changes of individual *configurations* but the inverse is not always true. There are *releases* where we can clearly see changes in individual *configurations* but not in the mean values. We have confirmed our assumption from the previous research question, that even when we can only see *substantial changes* in the *performance* but not in the *energy consumption*, changes are still present in both *non-functional properties*, but are not *substantial* for *energy consumption* in the mean values. For individual *configurations*, however, changes are often still *substantial* in both *non-functional properties*.

Combined over all case studies, out of the 73 investigated *releases*, we have identified 17 *prominent releases* in the previous research question (7 for both *non-functional properties* and another 10 only for the *performance*). In this research question, we found the in all 17 cases, changes are also reflected in individual *configurations*. In the results of the individual *configurations*, we found that out of the 10 *releases* which, in the mean values, only have *substantial changes* in the *performance*, 9 *releases* clearly show *substantial changes* in the *energy consumption* of individual *configurations*. We found three additional *releases* (two for *HSQLDB* and one for *PostgreSQL*, not considering *Apache httpd* here because of inconclusive results), with *substantial changes* in both *performance* and *energy consumption* of some *configurations*, that were not visible in the mean values. This results in a total number of 20 *prominent releases*, with all of them being *prominent* for the *performance* and all but one (95%) being *prominent* for the *energy consumption*.

Overall, we have identified *substantial changes* in 20 *releases* for the *performance* and 19 *releases* for the *energy consumption*. We were able to identify 85% of those for the *performance* and 37% for the *energy consumption* in the previous research question using the mean values. For the *performance*, none of the *prominent releases* have few (< 10%) *configurations* with *substantial changes* and 80% have many (> 50%) *configurations* with *substantial changes*. For the *energy consumption*, 16% of the *prominent releases* have few (< 10%) *configurations* with *substantial changes* and 37% have many (> 50%) *configurations* with *substantial changes*.

In the individual *configurations*, we observed a larger range of change values than in the mean values, with changes in the *performance* ranging from +28% to −83% and changes in the *energy consumption* ranging from +62% to −81%. (We observed all four extreme values in the *libvpx VP8* case study.)

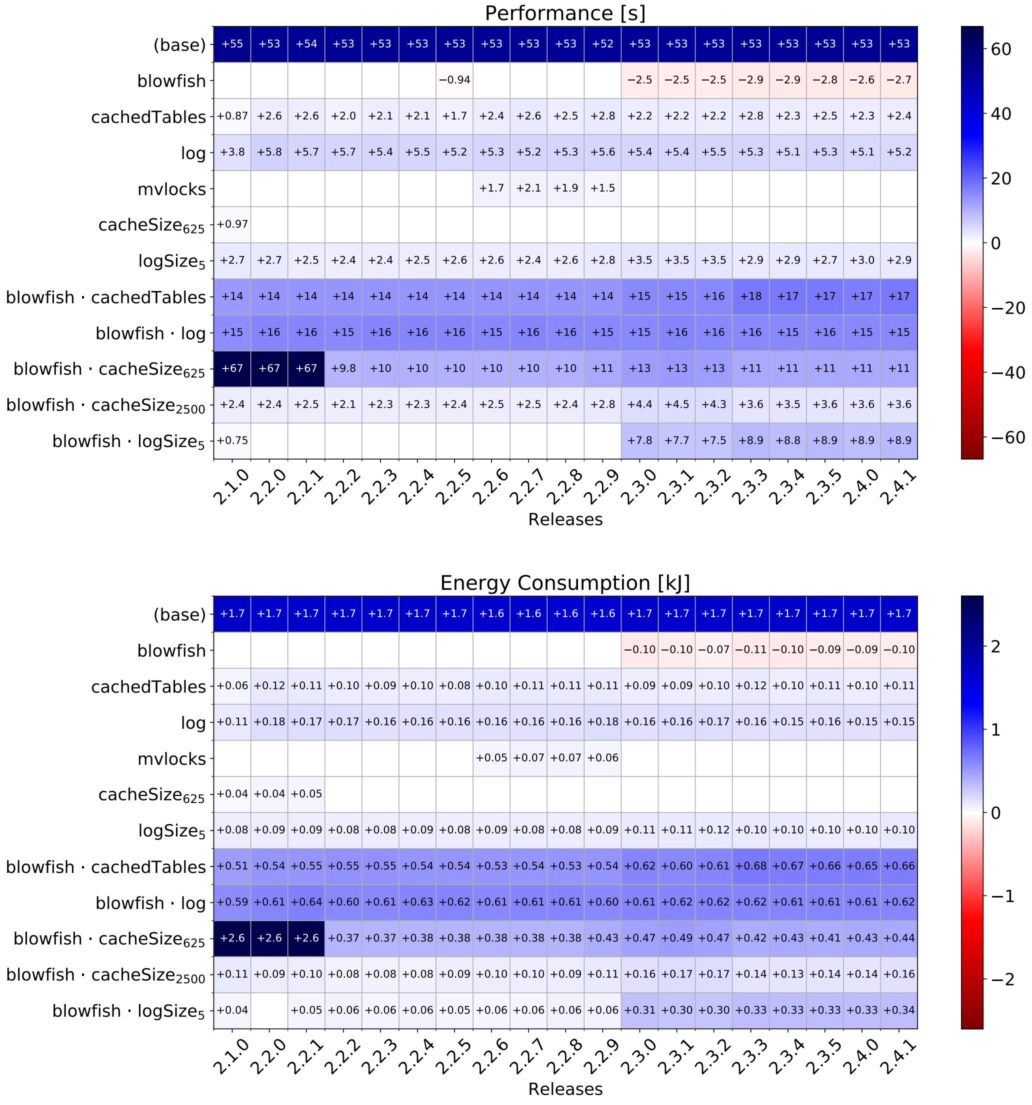
We found *substantial changes* in the *performance* of individual *configurations* for 27% of measured *releases* and in the *energy consumption* for 26% of measured *releases*. For *performance*, a majority (80%) of *releases* with *substantial changes* have changes in more than 50% of all *configurations*, whereas for *energy consumption*, only 37% of *releases* have *substantial changes* in more than 50% of all *configurations* and 16% have *substantial changes* in less than 10% of *configurations*. In almost all cases, changes impact both the *performance* and *energy consumption*, with 100% of *releases* with *substantial changes* in the *energy consumption* also having *substantial changes* in the *performance* and 95% of *releases* with *substantial changes* in the *performance* also having *substantial changes* in the *energy consumption*. Using individual *configuration* values, we confirmed all observations from the mean values and found three additional *releases* with *substantial changes*.

5.3 RQ1.3: Changes in Performance and Energy of Features

RQ1.3: Are changes in performance and energy consumption caused by individual features or feature interactions?

Results

For our third research question we increase the level of abstraction and investigate *performance-influence and energy-influence models* instead of the raw measurement



Performance [s]

	2.1.0	2.2.0	2.2.1	2.2.2	2.2.3	2.2.4	2.2.5	2.2.6	2.2.7	2.2.8	2.2.9	2.3.0	2.3.1	2.3.2	2.3.3	2.3.4	2.3.5	2.4.0	2.4.1
(base)	+55	+53	+54	+53	+53	+53	+53	+53	+53	+53	+52	+53	+53	+53	+53	+53	+53	+53	+53
blowfish							−0.94					−2.5	−2.5	−2.5	−2.9	−2.9	−2.8	−2.6	−2.7
cachedTables	+0.87	+2.6	+2.6	+2.0	+2.1	+2.1	+1.7	+2.4	+2.6	+2.5	+2.8	+2.2	+2.2	+2.2	+2.8	+2.3	+2.5	+2.3	+2.4
log	+3.8	+5.8	+5.7	+5.7	+5.4	+5.5	+5.2	+5.3	+5.2	+5.3	+5.6	+5.4	+5.4	+5.5	+5.3	+5.1	+5.3	+5.1	+5.2
mvlocks								+1.7	+2.1	+1.9	+1.5								
cacheSize$_{625}$	+0.97																		
logSize$_{5}$	+2.7	+2.7	+2.5	+2.4	+2.4	+2.5	+2.6	+2.6	+2.4	+2.6	+2.8	+3.5	+3.5	+3.5	+2.9	+2.9	+2.7	+3.0	+2.9
blowfish · cachedTables	+14	+14	+14	+14	+14	+14	+14	+14	+14	+14	+14	+15	+15	+16	+18	+17	+17	+17	+17
blowfish · log	+15	+16	+16	+15	+16	+16	+16	+15	+16	+16	+15	+15	+16	+16	+16	+15	+16	+15	+15
blowfish · cacheSize$_{625}$	+67	+67	+67	+9.8	+10	+10	+10	+10	+10	+10	+11	+13	+13	+13	+11	+11	+11	+11	+11
blowfish · cacheSize$_{2500}$	+2.4	+2.4	+2.5	+2.1	+2.3	+2.3	+2.4	+2.5	+2.5	+2.4	+2.8	+4.4	+4.5	+4.3	+3.6	+3.5	+3.6	+3.6	+3.6
blowfish · logSize$_{5}$	+0.75											+7.8	+7.7	+7.5	+8.9	+8.8	+8.9	+8.9	+8.9

Energy Consumption [kJ]

	2.1.0	2.2.0	2.2.1	2.2.2	2.2.3	2.2.4	2.2.5	2.2.6	2.2.7	2.2.8	2.2.9	2.3.0	2.3.1	2.3.2	2.3.3	2.3.4	2.3.5	2.4.0	2.4.1
(base)	+1.7	+1.7	+1.7	+1.7	+1.7	+1.7	+1.7	+1.6	+1.6	+1.6	+1.6	+1.7	+1.7	+1.7	+1.7	+1.7	+1.7	+1.7	+1.7
blowfish												−0.10	−0.10	−0.07	−0.11	−0.10	−0.09	−0.09	−0.10
cachedTables	+0.06	+0.12	+0.11	+0.10	+0.09	+0.10	+0.08	+0.10	+0.11	+0.11	+0.11	+0.09	+0.09	+0.10	+0.12	+0.10	+0.11	+0.10	+0.11
log	+0.11	+0.18	+0.17	+0.17	+0.16	+0.16	+0.16	+0.16	+0.16	+0.16	+0.18	+0.16	+0.16	+0.17	+0.16	+0.15	+0.16	+0.15	+0.15
mvlocks								+0.05	+0.07	+0.07	+0.06								
cacheSize$_{625}$	+0.04	+0.04	+0.05																
logSize$_{5}$	+0.08	+0.09	+0.09	+0.08	+0.08	+0.09	+0.08	+0.09	+0.08	+0.08	+0.09	+0.11	+0.11	+0.12	+0.10	+0.10	+0.10	+0.10	+0.10
blowfish · cachedTables	+0.51	+0.54	+0.55	+0.55	+0.55	+0.54	+0.54	+0.53	+0.54	+0.53	+0.54	+0.62	+0.60	+0.61	+0.68	+0.67	+0.66	+0.65	+0.66
blowfish · log	+0.59	+0.61	+0.64	+0.60	+0.61	+0.63	+0.62	+0.61	+0.61	+0.61	+0.60	+0.61	+0.62	+0.62	+0.62	+0.61	+0.61	+0.61	+0.62
blowfish · cacheSize$_{625}$	+2.6	+2.6	+2.6	+0.37	+0.37	+0.38	+0.38	+0.38	+0.38	+0.38	+0.43	+0.47	+0.49	+0.47	+0.42	+0.43	+0.41	+0.43	+0.44
blowfish · cacheSize$_{2500}$	+0.11	+0.09	+0.10	+0.08	+0.08	+0.08	+0.09	+0.10	+0.10	+0.09	+0.11	+0.16	+0.17	+0.17	+0.14	+0.13	+0.14	+0.14	+0.16
blowfish · logSize$_{5}$	+0.04		+0.05	+0.06	+0.06	+0.06	+0.05	+0.06	+0.06	+0.06	+0.06	+0.31	+0.30	+0.30	+0.33	+0.33	+0.33	+0.33	+0.34

Figure 5.8: *Performance-influence and energy-influence models* for the *HSQLDB* case study. Only values exceeding the threshold are shown.

results. As outlined in the section on methodology (Section 4.4.3), we generate *influence models* through a combination of the iterative and fitting approaches of *SPL Conqueror* and obtain one *influence model* for each combination of *non-functional property* and *release*. Instead of writing the models as formulas, we visualize them in a grid-like plot to allow for an easier comparison between different models.

In Figure 5.8, we visualize the models for the *HSQLDB* case study. The figure consists of two plots, one for the *performance-influence models* and one for the *energy-influence models*. The columns indicate the different *releases* and the rows denote the different terms that appear in the *influence models*, i.e. the *configuration options* and interactions between *configuration options*. The values in each of the cells are the respective factors from the *influence models*, so that each column of the plot corresponds to one *influence model*. The first row, '(base)', indicates the constant base value β_0 that is common for all *configurations*. To be able to more easily

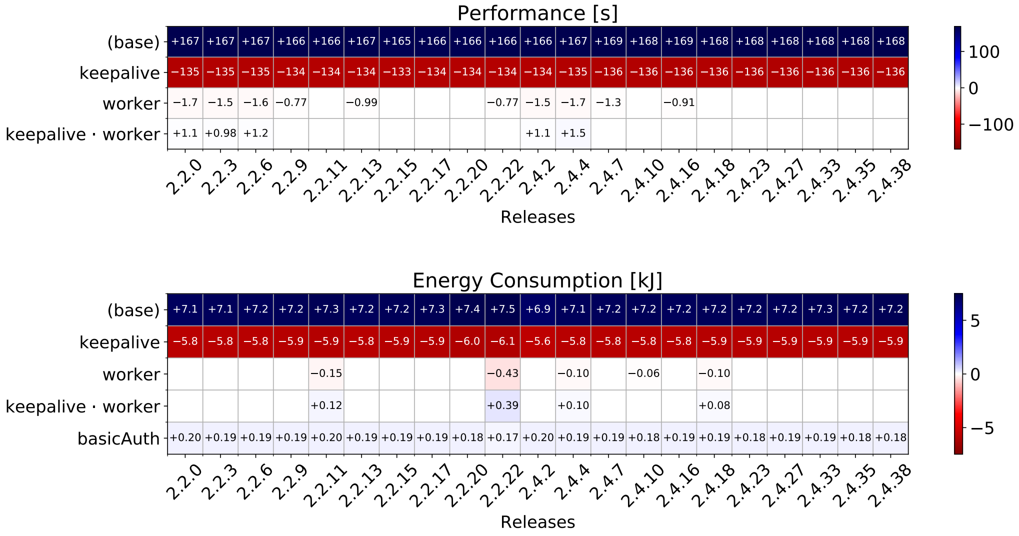


Figure 5.9: *Performance-influence and energy-influence models* for the *Apache httpd* case study. Only values exceeding the threshold are shown.

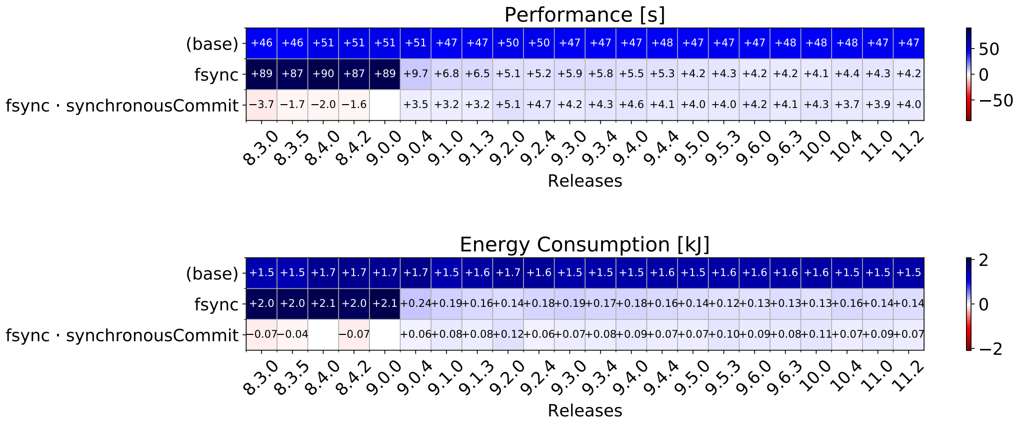


Figure 5.10: *Performance-influence and energy-influence models* for the *PostgreSQL* case study. Only values exceeding the threshold are shown.

visually compare the models, we additionally use colours for the different values. Blank cells indicate values that do not exceed each model's respective threshold, which is based on the model's learning error rate.

In the same way, we show the *performance-influence and energy-influence models* for the *Apache httpd* and *PostgreSQL* case studies in Figure 5.9 and Figure 5.10, respectively. Iterative learning only found a small number of terms for these two case studies, with four and five for the *performance-influence and energy-influence models*, respectively, of the *Apache httpd* case study, and three for both types of *influence models* of the *PostgreSQL* case study. We did not find any other influences upon manually examining the results, resulting in small *influence models* for these case studies.

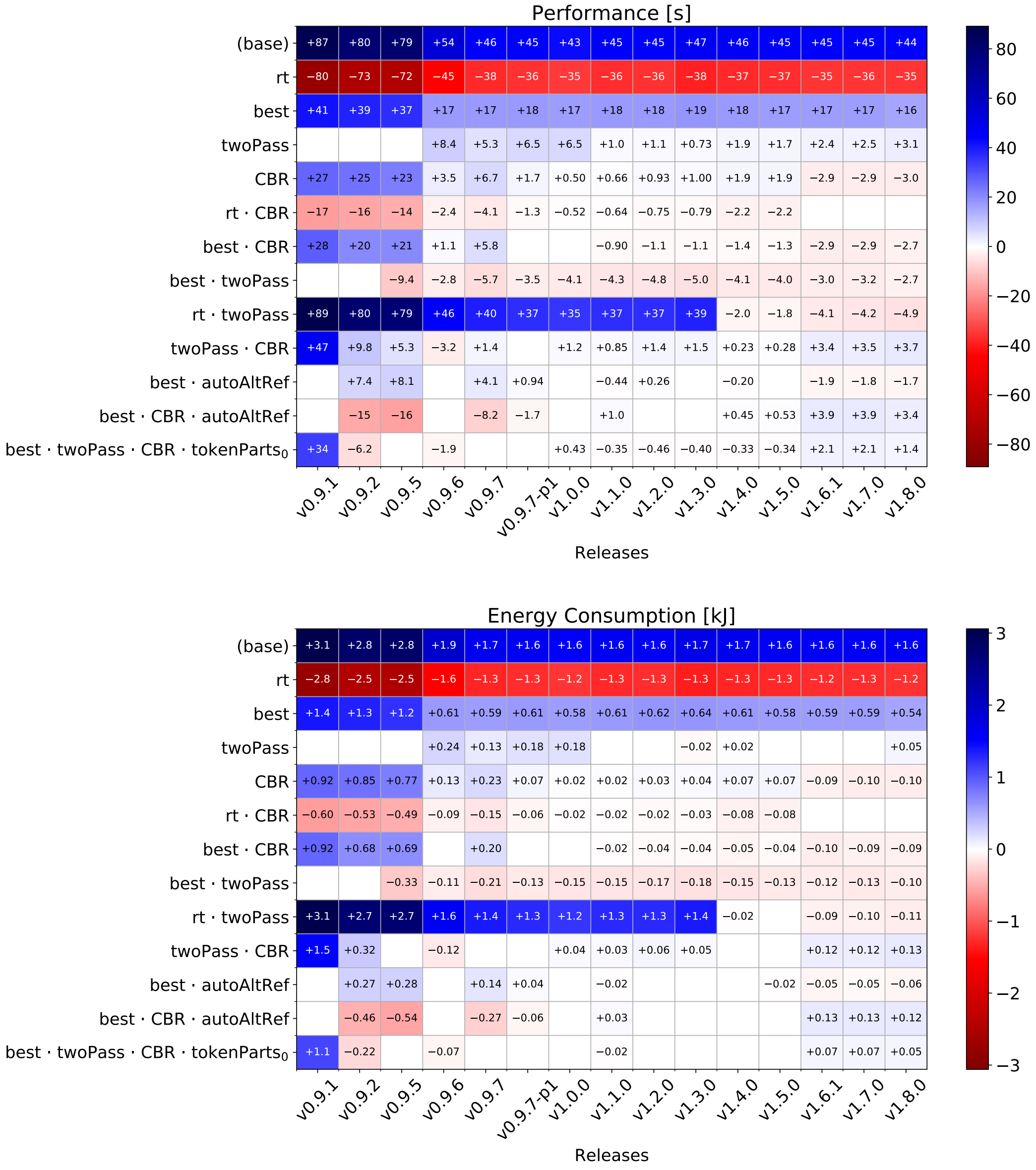


Figure 5.11: *Performance-influence and energy-influence models* for the *libvpx VP8* case study. Only values exceeding the threshold are shown. The models only represent *configurations* with *threads* = 1. The option 'constantBitrate' is abbreviated as 'CBR' for brevity. The last two influences were not found by iterative learning.

For the final case study, *libvpx VP8*, we have a numeric *configuration option* which defines the number of threads the encoder may use. This option has four values, 1, 2, 3 and 4, and in iteratively learned *influence models*, there is a large number of terms due to interactions of many *configuration options* with each of the values for the 'threads' option. Based on previous experience with a similar case study, we assumed that the behaviour of *performance* and *energy consumption* influences does not depend on the number of threads and thus generated four different *influence models* for each *non-functional property* and *release* of this case study, one for each value of the ‘threads’ option. We compared the resulting models and found no noticeable differences between the models for two, three and four threads. We found small differences between the models for one thread and the other models, with the main difference being that we found more terms in the models for one thread. Since the models for two, three and four threads are almost exactly the same, we do not show the models for three and four threads. Since the models for one and two threads are very similar, with the models for one thread containing slightly more information, we only show the models for one thread here, and include the models for two threads in Section A.3 of the appendix for reference. We show the model for a single thread, i.e. *influence models* learned on only those *configurations* where the *configuration option* ‘threads’ has a value of 1, in Figure 5.11.

For this case study, the iterative learning algorithm of *SPL Conqueror* only found influences of single *configuration option* or interactions between two *configuration options*. However, upon further investigation of the measurement results, we discovered influences of interactions between more than two *configuration options*, namely an interaction between the three options ‘best’, ‘constantBitrate’ and ‘autoAltRef’, as well as an interaction between the four terms[1] ‘best’, ‘constantBitrate’, ‘twoPass’ and ‘tokenParts’ with a value of 0. Since these interactions appear to have a noticeable influence as seen in the models generated with the fitting algorithm, we decided to keep these interactions in the models even though they were not found by the initial iterative model generation.

Discussion

In this research question, we use *influence models* to investigate whether changes in the *performance* and *energy consumption* are caused by specific *configuration options* or interactions of *configuration options*.

In the previous research questions, we identified a small increase in the *performance* in *release* 2.2.0 of *HSQLDB*. We have seen that this change is only visible in the *performance* but not the *energy consumption* and is caused by a large number of *configurations*. In the *influence models*, we can see small differences between the models for the previous *release*, 2.1.0, and the models of 2.2.0, but we cannot directly attribute this change to any specific model term.

The second change we identified for the *HSQLDB* case study was a larger negative change in both *performance* and *energy consumption* of version 2.2.2, which we found to be caused by only a small number of *configurations*. In the *influence models*, we can clearly see, that this change can be attributed to the feature interaction between blowfish encryption and a small cache size of 625 kilobytes.

Additionally, we have identified changes in the *releases* 2.3.0 and 2.3.3 that were not visible in the mean values but only in the values of individual *configurations*. In 2.3.0, we can see a change in the *influence models*, which appears to be related to blowfish encryption and several interactions therewith, in particular an interaction with a small log size of 5 megabytes. The change in 2.3.3 is not visible in the *influence models*.

[1] Actually between five terms, since this interaction only appears with *threads* = 1, but we are only focusing on those *configurations*.

There are no additional changes that are visible in the *influence models* that we had not previously identified in the mean values or the values of individual *configurations*.

A visual comparison between the plots of the *performance-influence models* and *energy-influence models* clearly shows that influences are very similar and also change in the same way for *performance* and *energy consumption*.

For the second case study, *Apache httpd*, we have not been able to identify any changes in the mean values and neither have seen any clear indications for changes in the individual *configurations*. In the same way, we cannot observe any noticeable changes in the *influence models*. We can see that the *influence models* for *performance* and *energy consumption* are very similar with only one exception: The *configuration option* 'basicAuth' does not appear to have a relevant influence on the *performance* but it has a small influence on the *energy consumption*.

For the *PostgreSQL* case study, we have even smaller *influence models* than for the previous case study. We have identified a large negative change in both *performance* and *energy consumption* with *release* 9.0.4, which is also clearly visible in the *performance-influence and energy-influence models* and can directly be linked to the 'fsync' *configuration option*. We observed a number of smaller changes in both the mean values, however, we have already seen in the values of the individual *configurations*, that these changes appear in almost all of the *configurations* and we can consequently assume that these changes cannot be attributed to a specific *configuration option*. We can confirm this assumption with the *influence models*, which show all of these changes in the base term β_0 ('(base)' in the figures).

As in the previous two case studies, we can see that the *influence models* for *performance* and *energy consumption* are virtually the same.

For the final case study, *libvpx VP8*, we identified a large number of changes in the mean values and have also seen a large number of changes in the individual *configurations*. Similarly, we can see several changes in the *performance-influence and energy-influence models*. The largest changes in the mean values were in *releases* v0.9.2 and v0.9.6. We can clearly see these changes in the *influence models*, with the change in v0.9.2 being caused by an interaction between 'twoPass' and 'CBR', as well as an interaction between those two *configuration options* and additionally 'best' and 'tokenParts' with a value of 0. The change in v0.9.6 is also clearly visible in the *influence models*, but cannot be attributed to any specific influences with almost all model terms showing changes in this *release*.

Additionally, we have identified changes in v0.9.7 and v1.4.0 in the mean values. The change in v0.9.7 is visible in the *influence models*, but not as clearly as the previous changes. It cannot be attributed to specific *configuration options*, which changes affecting most of the model terms. The change in v1.4.0 can clearly be attributed to an interaction between 'rt' and 'twoPass'. Upon further investigation of the results and consultation of the documentation, we discovered that this change is caused by an incorrect assumption we made when modelling the *configuration space* for this case study. In *releases* starting from v1.4.0, the two *configuration options* 'rt' and 'twoPass' can no longer be used in combination. When both options are specified, *libvpx VP8* simply ignores the 'twoPass' option.

Of the remaining smaller changes, those in v0.9.5 and 0.9.7p1 are also visible in the *influence models* but affect multiple unrelated terms. Other changes in previously identified *prominent releases* are not visible in the *influence models*.

In addition to the changes we have observed in the previous research question, we can observe an additional change in the *influence models* in v1.6.1, which appears to be of a small magnitude and affects almost all model terms, i.e. most *configuration options*. This change appears to have a positive influence on some *configurations* an a negative influence on other *configurations*. This explains why we did not see this change in the mean values, since positive and negative changes likely cancel each other out. These changes also appear to be of a relatively small magnitude, explaining why we could not identify them in the previous research question.

As for all previous case studies, we can see that the *performance-influence models* and *energy-influence models* are very similar and changes between *releases* are reflected in the same way in *influence models* for both *non-functional properties*.

For this case study, we manually added two feature interactions to the *influence models*, that *SPL Conqueror* did not discover during iterative learning. As we can see in the visualizations of the *influence models*, these interactions ($\text{best} \cdot \text{constantBitrate} \cdot \text{autoAltRef}$ and $\text{best} \cdot \text{twoPass} \cdot \text{constantBitrate} \cdot \text{tokenParts}_0$) have a noticeable influence in some *releases*, even larger than some of the iteratively learned influences, such as ‘twoPass’ and the interaction of ‘best’ and ‘twoPass’. This shows, that it can be worthwhile to complement the iterative learning of *SPL Conqueror* with additional strategies such as a manual investigation of measurement results to identify additional influences on a *non-functional property*.

Common for all four case studies we can see that influences and changes in influences behave very similarly for *performance* and *energy consumption*. Out of the 20 *releases* we found to have changes in *performance* or *energy consumption* in the previous research questions, 14 (70%) of changes are reflected in the *influence models*. In the models, we found one additional change that we did not identify previously. Out of the 15 changes we observed in the *influence models*, we could clearly attribute 5 (33%) changes to a specific *configuration option* or interaction between *configuration options*.

We found that 70% of the changes in *performance* and *energy consumption* we identified in previous research questions are reflected in *influence models*. We identified one additional change using the *influence models* that was not visible in the mean values or values of individual *configurations*. We could directly attribute 33% of the changes we observed in the *influence models* to a specific *configuration option* or interaction. Influences and changes of influences are consistent between *performance* and *energy consumption*.

5.4 RQ2.1: Changes in the Correlation

RQ2.1: Are there changes in the *correlation* between performance and energy consumption across releases?

Results

In RQ2, we are investigating the *correlation* between *performance* and *energy consumption* and changes in the *correlation* across *releases*. We do this by calculating the *Pearson correlation coefficient* between *performance* and *energy consumption* for all *configurations* of each *release*.

Release r		(all configurations)			$s = \neg$blowfish	
	ρ_r	interpretation	*subs. ch.*	$\rho_{r,s}$	interpretation	*subs. ch.*
2.1.0	0.99	very strong		0.39	weak	
2.2.0	0.99	very strong		0.67	strong	+
2.2.1	0.99	very strong		0.65	strong	
2.2.2	0.97	very strong		0.52	moderate	
2.2.3	0.98	very strong		0.60	moderate	
2.2.4	0.96	very strong		0.56	moderate	
2.2.5	0.97	very strong		0.57	moderate	
2.2.6	0.97	very strong		0.50	moderate	
2.2.7	0.97	very strong		0.62	strong	
2.2.8	0.97	very strong		0.58	moderate	
2.2.9	0.98	very strong		0.66	strong	
2.3.0	0.98	very strong		0.56	moderate	
2.3.1	0.98	very strong		0.58	moderate	
2.3.2	0.97	very strong		0.56	moderate	
2.3.3	0.98	very strong		0.55	moderate	
2.3.4	0.98	very strong		0.60	moderate	
2.3.5	0.98	very strong		0.61	strong	
2.4.0	0.98	very strong		0.59	moderate	
2.4.1	0.97	very strong		0.54	moderate	

Table 5.6: Overall correaltion for the *HSQLDB* case study. The 'subs. ch.' column indicates a positive (+) or negative (−) *substantial change* from the previous *release*. Correlation values are given for all *configurations* and for *configurations* without blowfish encryption.

For the first case study, *HSQLDB*, we considered in addition to the overall *correlation* of all *configurations* also the *correlation* of only *configurations* without blowfish encryption, since we have seen that this feature dominates *performance* and *energy consumption* and we also expect it to dominate the *correlation*. We show the *Pearson correlation coefficients* for this case study in Table 5.6. In the table, we show both the calculated *Pearson correlation coefficients* and our interpretation of these values. Additionally, we indicate whether there is a *substantial change* in the *correlation* from the previous *release*.

Release r		(all configurations)			$s = \neg$keepalive	
	ρ_r	interpretation	*subs. ch.*	$\rho_{r,s}$	interpretation	*subs. ch.*
2.2.0	0.99	very strong		0.06	none	
2.2.3	0.99	very strong		0.18	none	
2.2.6	0.99	very strong		0.03	none	
2.2.9	0.99	very strong		0.00	none	
2.2.11	0.99	very strong		0.10	none	
2.2.13	0.99	very strong		-0.08	none	
2.2.15	0.99	very strong		0.17	none	
2.2.17	0.99	very strong		0.24	weak	
2.2.20	0.99	very strong		0.20	weak	
2.2.22	0.98	very strong		0.35	weak	
2.4.2	0.99	very strong		0.01	none	$-$
2.4.4	1.00	perfect		0.26	weak	$+$
2.4.7	1.00	perfect		0.19	none	
2.4.10	1.00	perfect		0.26	weak	
2.4.16	1.00	perfect		0.16	none	
2.4.18	1.00	perfect		0.12	none	
2.4.23	1.00	perfect		0.33	weak	$+$
2.4.27	1.00	perfect		0.30	weak	
2.4.33	1.00	perfect		0.34	weak	
2.4.35	1.00	perfect		0.28	weak	
2.4.38	1.00	perfect		0.29	weak	

Table 5.7: Overall correaltion for the *Apache httpd* case study. The 'subs. ch.' column indicates a positive (+) or negative (−) *substantial change* from the previous *release*. Correlation values are given for all *configurations* and for *configurations* without keepalive connections.

For the *Apache httpd* case study, we show the *Pearson correlation coefficients* for all *configurations* and for *configurations* without the dominating *configuration option* 'keepalive' in Table 5.7.

For the third case study, *PostgreSQL*, we did not find an *configuration option* that dominates *performance* and *energy consumption* throughout all *releases*, so we only show the *Pearson correlation coefficients* for all *configurations* in Table 5.8.

For the *libvpx VP8* case study, we consider all *configurations* and additionally the *configurations* with a specific value for the 'threads' option, since we have already seen very similar behaviour across different values for that option in RQ1.3 and can see a similar behaviour here. For all *configurations*, we only observe very strong *Pearson correlation coefficients* ranging from 0.95 to 0.97, with no *substantial changes*. Considering only *configurations* with a single thread, all *releases* have a perfect *correlation* of 1.00. Similarly, *configurations* with a value of 2 for the 'threads' option have *Pearson correlation coefficients* ranging from 0.99 to 1.00. Just like in RQ1.3, there are no noticeable differences between the sets of *configurations* with the values 2, 3 and 4 for the 'threads' option.

Release r	ρ_r	interpretation	*subs. ch.*
8.3.0	0.91	very strong	
8.3.5	0.91	very strong	
8.4.0	0.95	very strong	
8.4.2	0.91	very strong	
9.0.0	0.95	very strong	
9.0.4	0.24	weak	$-$
9.1.0	0.53	moderate	$+$
9.1.3	0.10	none	$-$
9.2.0	0.45	moderate	$+$
9.2.4	0.28	weak	
9.3.0	0.48	moderate	$+$
9.3.4	0.41	moderate	
9.4.0	0.43	moderate	
9.4.4	0.22	weak	$-$
9.5.0	0.54	moderate	$+$
9.5.3	0.46	moderate	
9.6.0	0.37	weak	
9.6.3	0.52	moderate	
10.0	0.37	weak	
10.4	0.41	moderate	
11.0	0.34	weak	
11.2	0.03	none	$-$

Table 5.8: Overall correaltion for the *PostgreSQL* case study. The 'subs. ch.' column indicates a positive (+) or negative (−) *substantial change* from the previous *release*. Correlation values are given for all *configurations*.

Case study	Configurations	$\mathcal{R}^{\rho}_{\text{prom}}$
HSQLDB	(all)	$\emptyset$
	$\neg$blowfish	2.2.0
Apache httpd	(all)	$\emptyset$
	$\neg$keepalive	2.4.2, 2.4.4, 2.4.23
PostgreSQL	(all)	9.0.4, 9.1.0, 9.1.3, 9.2.0, 9.3.0, 9.4.4, 9.5.0, 11.2
libvpx VP8	(all)	$\emptyset$
	threads = 1	$\emptyset$
	threads = 2	$\emptyset$

Table 5.9: *Prominent releases* with respect to the *Pearson correlation coefficient* of each case study.

In Table 5.9, we list all *prominent releases* with respect to the *Pearson correlation coefficient* for each case study.

Discussion

For our first case study, *HSQLDB*, when investigating the *correlation* for all *configurations*, we can see that the *correlation* is generally very strong, ranging from 0.96 to 0.99, and there are no *substantial changes*. We found that this very strong *correlation* is primarily caused by the feature 'blowfish', which has a dominating influence on both the *performance* and the *energy consumption*. When considering only *configurations* without this option, we can see a lower *correlation* ranging from a weak *correlation* of 0.39 to a strong *correlation* of 0.67. There is one *substantial change*, with a jump from a weak to a strong *correlation* in *release* 2.2.0. This change in the *correlation* coincides with the only change that we observed across all case studies, that only affected the *performance* but not the *energy consumption*, not only in the mean values but also in the values of individual *configurations*, where for all other changes we observed, *performance* and *energy consumption* where affected alike. This connection makes sense, since a change in only one of the *non-functional properties* necessarily affects the *correlation*. However, we found no indication for a reason for this behaviour in version 2.2.0 of *HSQLDB*. There are no further changes in the *correlation* of *HSQLDB*.

For the second case study, *Apache httpd*, when investigating all *configurations*, we can observe a very strong or even perfect *correlation* ranging from 0.98 to 1.00. Similar to blowfish encryption in *HSQLDB*, there is also a dominating *configuration option* in this case study, namely the 'keepalive' option. When calculating the *correlation* values only for *configurations* without the 'keepalive' option, we can see a much lower *correlation*, ranging from no *correlation* at all with a value of 0.00 to a weak *correlation* with a maximum value of 0.35. According to our definition of *substantial changes*, there are three *substantial changes* in the *correlation* of *Apache httpd* without 'keepalive', with one change from a weak *correlation* to no *correlation* in version 2.4.2, a change back to a weak *correlation* in the next *release*, 2.4.4, and a change from no *correlation* to a weak *correlation* in 2.4.23. However, all of these changes are only between no *correlation* and weak *correlation*, which, considering the high level of noise we have observed in previous research question for this case study, are, albeit *substantial* according to our definition, not in any way relevant for our evaluation.

In the third case study, *PostgreSQL*, we can see a wide range of *correlation* values, ranging from no *correlation* with a value of 0.03 to a very strong *correlation* of 0.95. We can also see a number of *substantial changes*, most notably a drop from a very strong to a weak *correlation* in version 9.0.4 and a change from a moderate to no *correlation* in 9.1.3 followed by a change back to a moderate *correlation* in the subsequent *release*, 9.2.0. Additionally, there are a number of smaller changes between weak and moderate or between weak and no *correlation*, which, similar to the situation in the *Apache httpd* case study, are *substantial* according to the definition, but are not particularly relevant for this evaluation.

The drop in the *correlation* in version 9.0.4 coincides with the large change in *performance* and *energy consumption* we have observed in this *release* in previous research questions. This was the only *release* in which we observed considerably different magnitudes of changes for *performance* and *energy consumption*, with the *performance* dropping by 40% and the *energy consumption* dropping only by 32%. The different changes in *performance* and *energy consumption* explain the change in the *correlation* and we have already found an explanation for the different changes in a previous research question, with this change being caused by the 'fsync' *configuration option*, which is a feature that introduces a delay with a direct impact on the *performance* but only a comparably small influence on the *energy consumption*.

The drop in the *correlation* in *release* 9.1.3 does not coincide with any changes in the *performance* or *energy consumption* and the change in the *correlation* of the following *release*, 9.2.0, corresponds to only a small change in the *performance* and *energy consumption*. The reason for this might be, that even large and *substantial* changes in the *correlation* like these can be caused by noise and the inaccuracy of the *correlation*.

In the *libvpx VP8* case study, we observed a very strong *correlation* throughout all *releases* and could not identify a dominating *configuration option*, the exclusion of which might lower the *correlation*. Since the *correlation* is very strong in all *releases*, there are no *substantial changes*.

To compare the combined results of all case studies with our findings from the previous research questions, we consider for the *HSQLDB* case study the *correlation* of *configurations* without 'blowfish', for the *Apache httpd* case study the *correlation* of *configurations* without 'keepalive', for *PostgreSQL* the *correlation* of all *configurations* and for *libvpx VP8* the *correlation* of *configurations* with a single thread. Out of the 20 *releases* we have identified to have *substantial changes* in the *performance* or *energy consumption*, we have 5 *releases* (25%) with a *substantial change* in the *correlation* between *performance* and *energy consumption*. In the inverse direction, out of the 12 *releases* we found to have a *substantial change* in the *correlation*, 5 *releases* (42%) have shown *substantial changes* in the *performance* or *energy consumption*. Overall, we have observed *substantial changes* in 12 (16%) of all 73 measured *releases*.

Overall, we can see that there is no clear connection between changes in the *correlation* and changes in the *performance* and *energy consumption*. However, the most apparent changes in the *correlation* are reflected by unusual changes in *performance* and *energy consumption* and vice versa. We have found *substantial changes* in the *correlation* for both the one *release* with a change in only one of the *non-functional properties* and the one *release* with considerably differing changes in the two *non-functional properties*.

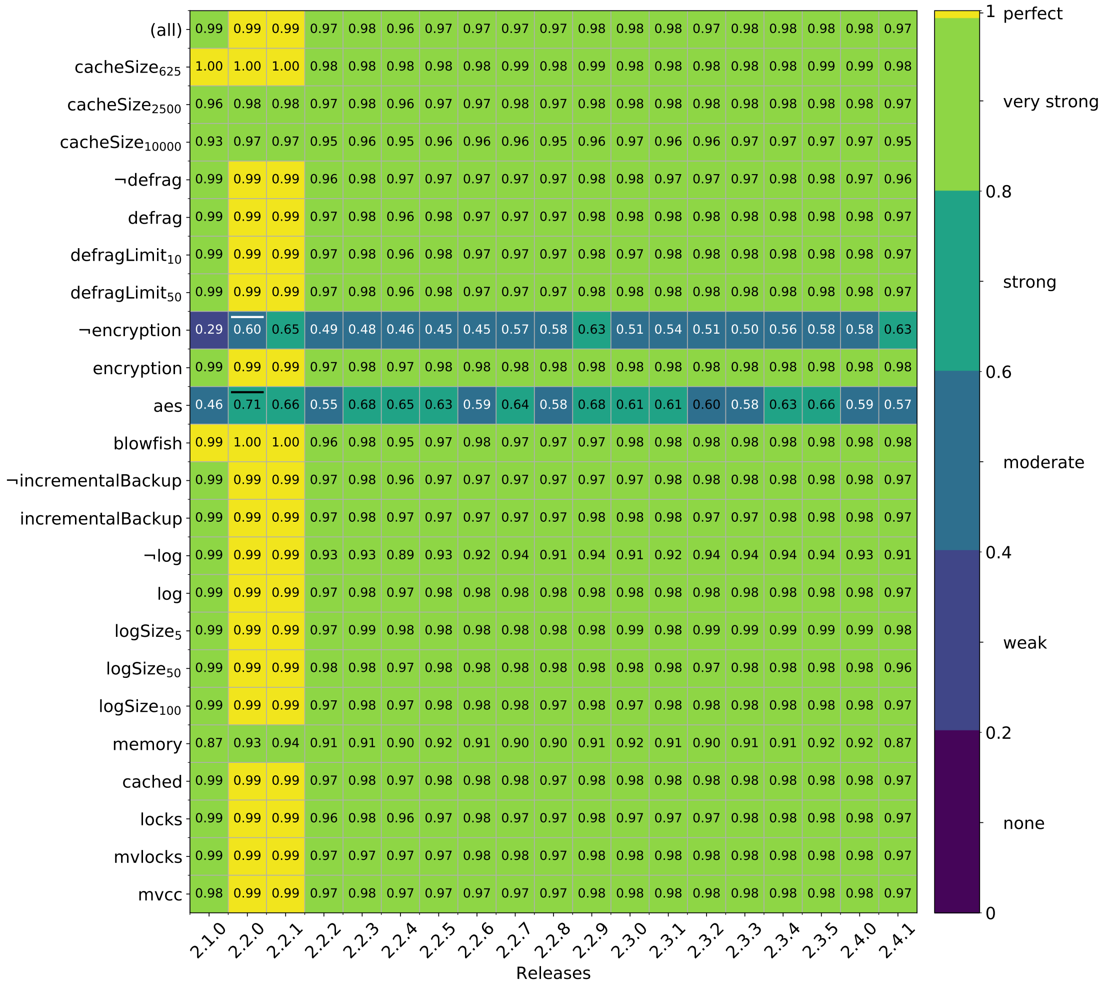
We found *substantial changes* in the *correlation* between *performance* and *energy consumption* in 16% of measured *releases*. 25% of *releases* with changes in *performance* and *energy consumption* also show changes in the *correlation* and 42% of *releases* with changes in the *correlation* also have changes in the *performance* and *energy consumption*.

5.5 RQ2.2: Changes in the Correlation of Features

RQ2.2: Are changes in the *correlation* between performance and energy consumption caused by specific individual features?

Results

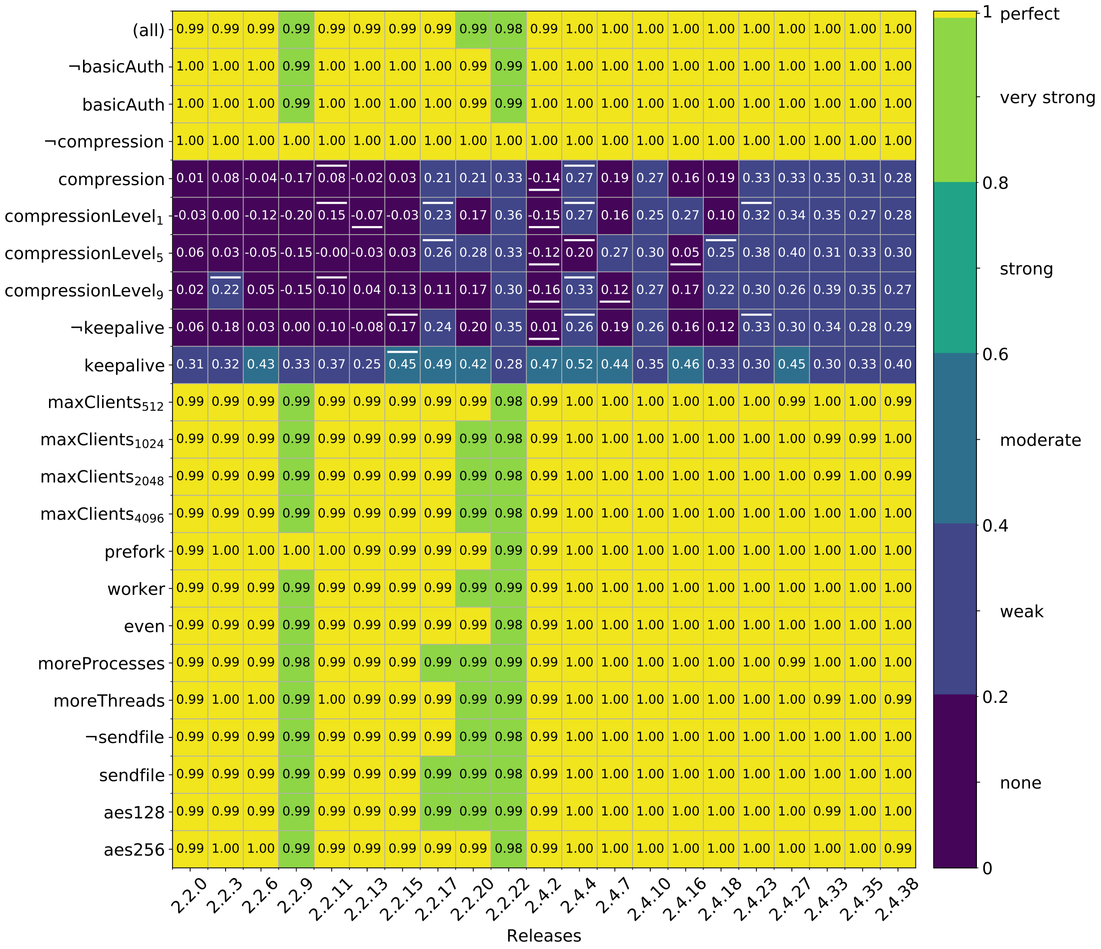
In our final research question, we again investigate the *correlation* between *performance* and *energy consumption*, increasing the granularity from the previous research question to consider the *correlation* for different subsets of the *configuration space*.



	2.1.0	2.2.0	2.2.1	2.2.2	2.2.3	2.2.4	2.2.5	2.2.6	2.2.7	2.2.8	2.2.9	2.3.0	2.3.1	2.3.2	2.3.3	2.3.4	2.3.5	2.4.0	2.4.1
(all)	0.99	0.99	0.99	0.97	0.98	0.96	0.97	0.97	0.97	0.97	0.98	0.98	0.98	0.97	0.98	0.98	0.98	0.98	0.97
cacheSize_{625}	1.00	1.00	1.00	0.98	0.98	0.98	0.98	0.98	0.99	0.98	0.99	0.98	0.98	0.98	0.98	0.98	0.99	0.99	0.98
cacheSize_{2500}	0.96	0.98	0.98	0.97	0.98	0.96	0.97	0.97	0.98	0.97	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.97
cacheSize_{10000}	0.93	0.97	0.97	0.95	0.96	0.95	0.96	0.96	0.96	0.95	0.96	0.97	0.96	0.96	0.97	0.97	0.97	0.97	0.95
¬defrag	0.99	0.99	0.99	0.96	0.98	0.97	0.97	0.97	0.97	0.97	0.98	0.98	0.97	0.97	0.97	0.98	0.98	0.97	0.96
defrag	0.99	0.99	0.99	0.97	0.98	0.96	0.98	0.97	0.97	0.97	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.97
defragLimit_{10}	0.99	0.99	0.99	0.97	0.98	0.96	0.98	0.97	0.97	0.97	0.98	0.97	0.98	0.98	0.98	0.98	0.98	0.98	0.97
defragLimit_{50}	0.99	0.99	0.99	0.97	0.98	0.96	0.98	0.97	0.97	0.97	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.97
¬encryption	0.29	$\overline{0.60}$	0.65	0.49	0.48	0.46	0.45	0.45	0.57	0.58	0.63	0.51	0.54	0.51	0.50	0.56	0.58	0.58	0.63
encryption	0.99	0.99	0.99	0.97	0.98	0.97	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98
aes	0.46	$\overline{0.71}$	0.66	0.55	0.68	0.65	0.63	0.59	0.64	0.58	0.68	0.61	0.61	0.60	0.58	0.63	0.66	0.59	0.57
blowfish	0.99	1.00	1.00	0.96	0.98	0.95	0.97	0.98	0.97	0.97	0.97	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98
¬incrementalBackup	0.99	0.99	0.99	0.97	0.98	0.96	0.97	0.97	0.97	0.97	0.97	0.97	0.98	0.98	0.98	0.98	0.98	0.98	0.97
incrementalBackup	0.99	0.99	0.99	0.97	0.98	0.97	0.97	0.97	0.97	0.97	0.98	0.98	0.98	0.97	0.97	0.98	0.98	0.98	0.97
¬log	0.99	0.99	0.99	0.93	0.93	0.89	0.93	0.92	0.94	0.91	0.94	0.91	0.92	0.94	0.94	0.94	0.94	0.93	0.91
log	0.99	0.99	0.99	0.97	0.98	0.97	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.97
logSize_{5}	0.99	0.99	0.99	0.97	0.99	0.98	0.98	0.98	0.98	0.98	0.98	0.99	0.98	0.99	0.99	0.99	0.99	0.99	0.98
logSize_{50}	0.99	0.99	0.99	0.98	0.98	0.97	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.97	0.98	0.98	0.98	0.98	0.96
logSize_{100}	0.99	0.99	0.99	0.97	0.98	0.97	0.98	0.98	0.98	0.97	0.98	0.97	0.98	0.98	0.98	0.98	0.98	0.98	0.97
memory	0.87	0.93	0.94	0.91	0.91	0.90	0.92	0.91	0.90	0.90	0.91	0.92	0.91	0.90	0.91	0.91	0.92	0.92	0.87
cached	0.99	0.99	0.99	0.97	0.98	0.97	0.98	0.98	0.98	0.97	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.97
locks	0.99	0.99	0.99	0.96	0.98	0.96	0.97	0.98	0.97	0.97	0.98	0.97	0.97	0.97	0.98	0.98	0.98	0.98	0.97
mvlocks	0.99	0.99	0.99	0.97	0.97	0.97	0.97	0.98	0.98	0.97	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.97
mvcc	0.98	0.99	0.99	0.97	0.98	0.97	0.97	0.97	0.97	0.97	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.97

Figure 5.12: *Pearson correlation coefficient* for subsets of the *configuration space* for each *release* of the *HSQLDB* case study. A bar over a correlation value indicates a positive *substantial change* from the previous *release*, a bar under a value indicates a negative *substantial change*.

In Figure 5.12, we show the *Pearson correlation coefficients* between *performance* and *energy consumption* for different subsets of the *configuration space* of the first



	2.2.0	2.2.3	2.2.6	2.2.9	2.2.11	2.2.13	2.2.15	2.2.17	2.2.20	2.2.22	2.4.2	2.4.4	2.4.7	2.4.10	2.4.16	2.4.18	2.4.23	2.4.27	2.4.33	2.4.35	2.4.38
(all)	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.98	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
¬basicAuth	1.00	1.00	1.00	0.99	1.00	1.00	1.00	1.00	0.99	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
basicAuth	1.00	1.00	1.00	0.99	1.00	1.00	1.00	1.00	0.99	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
¬compression	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
compression	0.01	0.08	-0.04	-0.17	$\overline{0.08}$	-0.02	0.03	0.21	0.21	0.33	$\underline{-0.14}$	$\overline{0.27}$	0.19	0.27	0.16	0.19	0.33	0.33	0.35	0.31	0.28
compressionLevel$_1$	-0.03	0.00	-0.12	-0.20	$\overline{0.15}$	$\underline{-0.07}$	-0.03	$\overline{0.23}$	0.17	0.36	$\underline{-0.15}$	$\overline{0.27}$	0.16	0.25	0.27	0.10	$\overline{0.32}$	0.34	0.35	0.27	0.28
compressionLevel$_5$	0.06	0.03	-0.05	-0.15	-0.00	-0.03	0.03	$\overline{0.26}$	0.28	0.33	$\underline{-0.12}$	$\overline{0.20}$	0.27	0.30	$\underline{0.05}$	$\overline{0.25}$	0.38	0.40	0.31	0.33	0.30
compressionLevel$_9$	0.02	$\overline{0.22}$	0.05	-0.15	$\overline{0.10}$	0.04	0.13	0.11	0.17	0.30	$\underline{-0.16}$	$\overline{0.33}$	$\underline{0.12}$	0.27	0.17	0.22	0.30	0.26	0.39	0.35	0.27
¬keepalive	0.06	0.18	0.03	0.00	0.10	-0.08	$\overline{0.17}$	0.24	0.20	0.35	$\underline{0.01}$	$\overline{0.26}$	0.19	0.26	0.16	0.12	$\overline{0.33}$	0.30	0.34	0.28	0.29
keepalive	0.31	0.32	0.43	0.33	0.37	0.25	$\overline{0.45}$	0.49	0.42	0.28	0.47	0.52	0.44	0.35	0.46	0.33	0.30	0.45	0.30	0.33	0.40
maxClients$_{512}$	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.98	0.99	1.00	1.00	1.00	1.00	1.00	1.00	0.99	1.00	1.00	0.99
maxClients$_{1024}$	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.98	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.99	1.00
maxClients$_{2048}$	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.98	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	1.00	0.99
maxClients$_{4096}$	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.98	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
prefork	0.99	1.00	1.00	1.00	1.00	0.99	0.99	0.99	0.99	0.99	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
worker	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
even	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.98	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
moreProcesses	0.99	0.99	0.99	0.98	0.99	0.99	0.99	0.99	0.99	0.99	0.99	1.00	1.00	1.00	1.00	1.00	1.00	0.99	1.00	1.00	1.00
moreThreads	0.99	1.00	1.00	0.99	1.00	0.99	0.99	0.99	0.99	0.99	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	1.00	0.99
¬sendfile	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.98	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
sendfile	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.98	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
aes128	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	1.00	1.00
aes256	0.99	1.00	1.00	0.99	0.99	0.99	0.99	0.99	0.99	0.98	0.99	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99

Figure 5.13: *Pearson correlation coefficient* for subsets of the *configuration space* for each *release* of the *Apache httpd* case study. A bar over a correlation value indicates a positive *substantial change* from the previous *release*, a bar under a value indicates a negative *substantial change*.

case study, *HSQLDB*. Columns indicate the different *releases* and rows indicate the *constraints* used to define subsets of the *configuration space*, one for each binary option, for each alternative from alternative groups and for each value of numeric options. The first row, '(all)' indicates the full *configuration space* with no *constraints*. Values in the cells are the *Pearson correlation coefficients* with colours indicating the different levels of *correlation* according to our interpretation. *Substantial changes* are highlighted with a bar over or under the value for positive and negative changes, respectively.

Following the same schema, we show the *Pearson correlation coefficients* for the *Apache httpd* and *PostgreSQL* case studies in Figure 5.13 and Figure 5.14, respectively.

For the final case study, *libvpx VP8*, when considering the whole *configuration space*, the *Pearson correlation coefficients* range from a very strong *correlation* of 0.92 to a perfect *correlation* for all subsets of *configurations* and for all *releases*. Instead, we consider, just like in the previous research questions, only the *configurations* with

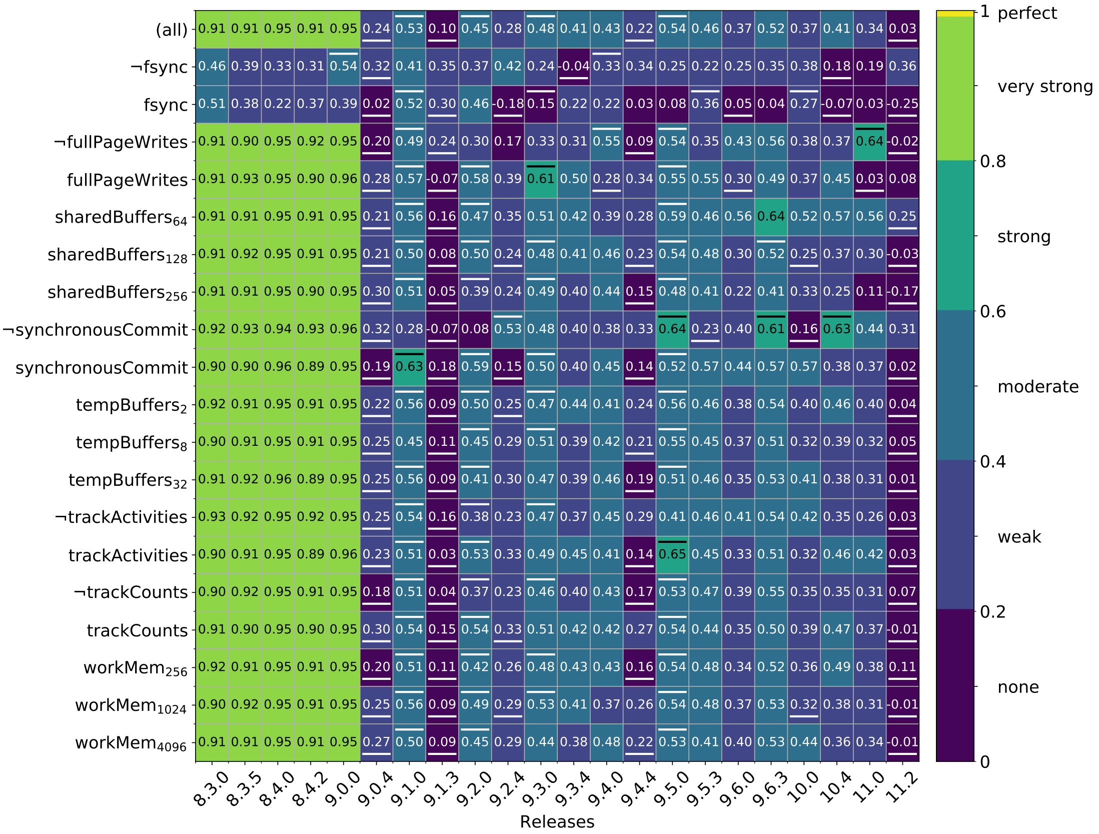


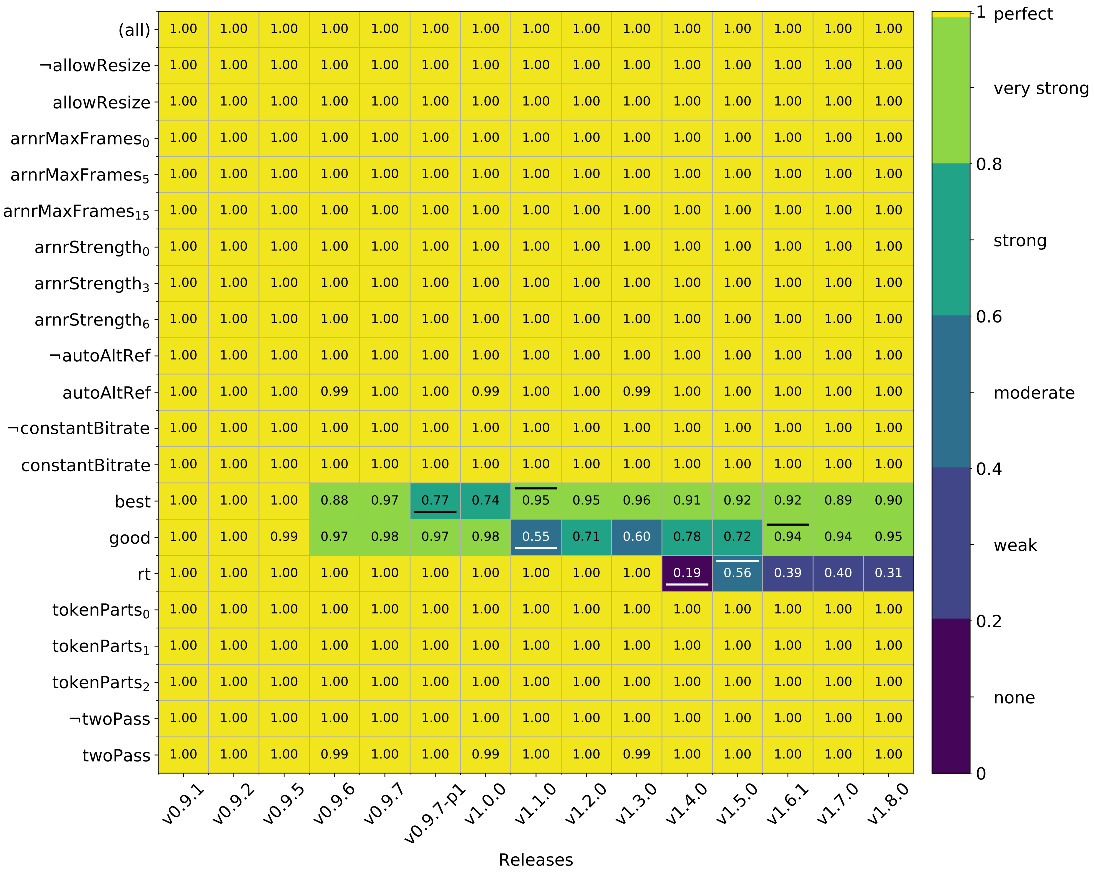
Figure 5.14: *Pearson correlation coefficient* for subsets of the *configuration space* for each *release* of the *PostgreSQL* case study. A bar over a correlation value indicates a positive *substantial change* from the previous *release*, a bar under a value indicates a negative *substantial change*.

a specific value for the 'threads' option and take subsets from this set. We show the *Pearson correlation coefficients* for subsets of the set of *configurations* with a value of 1 for the 'threads' option in Figure 5.15 and include the plot for value 2 in Section A.3 of the appendix for reference.

Discussion

For the final research question, we increase the granularity from the previous research question and investigate the *correlation* between *performance* and *energy consumption* for different subsets of the *configuration space* to identify different *correlation* behaviour of the different *configuration options*.

For the *HSQLDB* case study, we observe a very strong *correlation* for all *configurations* as well as for most subsets of the *configuration space*, with only two exceptions: *configurations* without 'encryption' and *configurations* with 'aes'. Those two sets have in common the absence of the feature 'blowfish', since 'blowfish' and 'aes' form an alternative group with 'encryption' as its root. There are only two *substantial changes* in the *correlation*, both in the subsets without 'blowfish' and both in *release* 2.2.0. This is the same change that we found in the previous research question, where we considered the *correlation* for all *configurations* without the dominating



	v0.9.1	v0.9.2	v0.9.5	v0.9.6	v0.9.7	v0.9.7-p1	v1.0.0	v1.1.0	v1.2.0	v1.3.0	v1.4.0	v1.5.0	v1.6.1	v1.7.0	v1.8.0
(all)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
¬allowResize	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
allowResize	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
arnrMaxFrames$_0$	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
arnrMaxFrames$_5$	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
arnrMaxFrames$_{15}$	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
arnrStrength$_0$	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
arnrStrength$_3$	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
arnrStrength$_6$	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
¬autoAltRef	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
autoAltRef	1.00	1.00	1.00	0.99	1.00	1.00	0.99	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00
¬constantBitrate	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
constantBitrate	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
best	1.00	1.00	1.00	0.88	0.97	0.77	0.74	0.95	0.95	0.96	0.91	0.92	0.92	0.89	0.90
good	1.00	1.00	0.99	0.97	0.98	0.97	0.98	0.55	0.71	0.60	0.78	0.72	0.94	0.94	0.95
rt	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.19	0.56	0.39	0.40	0.31
tokenParts$_0$	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
tokenParts$_1$	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
tokenParts$_2$	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
¬twoPass	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
twoPass	1.00	1.00	1.00	0.99	1.00	1.00	0.99	1.00	1.00	0.99	1.00	1.00	1.00	1.00	1.00

Figure 5.15: *Pearson correlation coefficient* for subsets of the *configuration space* for each *release* of the *libvpx VP8* case study. A bar over a correlation value indicates a positive *substantial change* from the previous *release*, a bar under a value indicates a negative *substantial change*. Only *configurations* with *threads* = 1 are considered.

feature 'blowfish'. This is, however, not a change that is caused by 'blowfish' but rather a change that is hidden by the dominance of the *configuration option*.

Similar results to the previous research question, we can also see for the *Apache httpd* case study. There is a very strong *correlation* in all subsets of the *configuration space* expect for those with 'compression', a 'compressionLevel' (which implies 'compression') and a specific value for 'keepalive'. Since 'compression' and 'keepalive' are mutually exclusive, all *configurations* with 'compression' are without 'keepalive'. While neither *configurations* without 'keepalive' nor those with 'keepalive' show a strong *correlation*, the dominating influence of this *configuration option* causes a strong *correlation* for sets containing *configurations* both with and without 'keepalive'. Within the sets with weaker *correlation*, *substantial changes* are frequent and noise is the most likely cause.

The first five *releases* of the *PostgreSQL* case study behave like the previous two case studies, showing a very strong *correlation* except in the sets of *configurations* with or without 'fsync', because the 'fsync' option dominates *performance* and *energy consumption* in those *releases*. Starting with *release* 9.0.4, there is a much weaker *correlation*, consistently across all subsets of the *configuration space*. There are many *substantial changes*, some of which will be caused by noise. However, it is clear that in some *releases* all or almost all subsets of the *configuration space* show the same *substantial change* in the *correlation*. These are the same *releases* in which we observed *substantial changes* in the *correlation* of all *configurations*. The fact that the *correlation* shows a similar behaviour regarding the *substantial changes* for all subsets of the *configuration space* shows us, that these changes are not caused by any specific *configuration option*, but rather affect all *configuration options* and all *configurations* alike. The change in 9.0.4 is the only exception from this behaviour, with this change being much larger for sets of *configurations* with mixed values for 'fysnc', since, as we previously discovered, this change is caused by that particular *configuration option*.

In the last case study, *libvpx VP8*, we focused on those *configurations* with threads = 1. We can see a perfect *correlation* in most subsets of the *configuration space* throughout all *releases*. Only the sets of *configurations* with a specific *configuration option* from the alternative group with 'best', 'good' and 'rt' have a weaker *correlation* for some *releases*. *Configurations* with the option 'best' have a *correlation* in v0.9.7-p1 and v1.0.0 that is *substantially* lower than that of other *releases*. *Configurations* with the option 'good' have *substantially* lower *correlation* in *releases* v1.1.0 to v1.5.0. Finally, the *correlation* for *configurations* with 'rt' is *substantially* lower starting with *release* v1.4.0. This change is related to the missing *constraint* in our *feature model* for this case study which we described in a previous research question.

Out of the twelve *prominent releases* with respect to the *correlation*, we only found one instance of a change in the overall *correlation* being caused by a specific *configuration option*, namely 'fsync' in *release* 9.0.4 of *PostgreSQL*. This coincides with the only instance of a *prominent release* with respect to the *correlation* having a change in the *performance* and *energy consumption*, that we could clearly attribute to a specific *configuration option* using the *influence models*. There are four more *releases* which have *substantial changes* both in *performance* and *energy consumption*, and also in the *correlation*. For all of those, we have seen changes in the *influence models* but could not attribute the changes to any specific *configuration option*.

We only found a single instance of a change in the overall *correlation* between *performance* and *energy consumption* that we could directly attribute to a specific *configuration option*. A single observation is not enough to answer the question of whether there are changes in the *correlation* that are caused by individual *configuration options*, with confidence.

6. Validity

In this chapter, we discuss the validity of our results. First we consider the internal validity, i.e. limitations and potential sources of errors in our measurements. Then we discuss the external validity, i.e. the generalizability of our findings.

6.1 Internal Validity

In this section, we list potential sources of errors in our measurements and restrictions in our experimental setup, which may lead to inaccurate or invalid results. For each of these, we describe the measures we have taken to mitigate these threats to the internal validity of our findings.

Repetitions and Deviation

In the following paragraphs we list specific threats and measures, but those measures cannot cover all potential sources of errors. For this reason, we measure five repetitions of each *configuration* and take the mean value of all repetitions. Additionally, we determine the relative standard deviation of the repetitions. The relative standard deviation is calculated as the absolute standard deviation divided by the mean value. If it exceeds 10% for a *configuration*, we discard the results and repeat the measurements for this *configuration*.

CPU

Modern CPUs have several complex features that aim at improving performance or reducing power consumption under specific conditions. These conditions are virtually impossible to predict and consequently have unpredictable influences on the measurements. However, these CPU features are used in practice and disabling them for the measurements would represent an unrealistic scenario. For this reason we decided to perform our measurements without disabling these features, assuming that any unpredictable influences would either be consistent across different measurements, or otherwise influences would be diminished by the averaging over multiple repeated measurements.

For software that is executed on multiple CPU cores, the CPU may dynamically and frequently change which cores the software is executed on. This can lead to CPU cores frequently switching between idle and active states which in turn can impact the *energy consumption* and *performance* in an unpredictable way. For the *libvpx VP8* case study, which – depending on the value of the 'threads' option – runs only on some of the cores of the CPU, we assigned specific cores to the process using the `taskset` command to reduce this impact. For other case studies, we did not restrict which cores should be used for a more realistic situation.

Background Software

Software running in the background during the measurements, can influence the measured *performance* and *energy consumption*. Since we use a minimal operating system installation, we can expect there to be little to no influence from background applications most of the time. In the rare case that there is an impact from software running in the background (e.g. automatic updates), it will only appear in one of the repetitions and we will detect it through a high deviation.

As stated in the section describing our experimental setup (Section 4.3), we use different nodes for servers and clients to avoid clients influencing servers and we use a separate host to query and process power consumption values.

PDUs

The PDUs are another potential source of measurement errors for the *energy consumption*. Measurements could yield different results when different sockets of the PDU are used or could be affected by external influences that are not stable over time. Additionally, simultaneous measurements for different cluster nodes connected to the same PDU may influence one another.

In our previous work [Wer17], we had used the same PDUs as in the current experimental setup. In that work, we conducted thorough tests to investigate whether the PDUs are suitable for our measurements. In the following, we list the properties we considered as relevant for accurate measurements with the PDUs and summarize our respective findings:

Offsets: We found that there are constant offsets in the measured values between different sockets of the PDUs. We can simply subtract these offsets to obtain comparable results.

Consistency: We found that measuring a constant load yields constant measurement results.

Comparability: We compared measurements with a different meter and with a fixed load, and found that measurements yield the correct results.

Repeatability: We found that repeated measurements yield equal results.

Isolation: There was no indication for multiple simultaneous measurements influencing the results of on another.

Network

For case studies with a client–server setup, the network connection between the cluster nodes may be a limiting factor for the *performance*. In our experimental setup, we have to assume that there is an influence on the *performance* from the network, but we are confident that it is small, since the CPU load on the server is on average higher than 10%[1] for all *configurations*. This indicates, that the server is not only waiting for the network and we measure at least partially the *performance* of the server.

On the cluster nodes, all user files are not stored on a local disk but in a network file system. Reading the measured software binaries and benchmark input files from the network file systems could introduce unwanted delays and impact the *performance*. To avoid this, we copy all files that are used during the measurements to the local disk (SSD) of the respective cluster node before starting a measurement.

Warm-Up

Software running – or not running – on a cluster node directly before the measurements can influence the results. In particular the temperature of the CPU can have an influence on the *performance* as stated by Mytkowicz et al. [MDHS09]. To ensure equal initial conditions for all measurements, we include a warm-up phase for the CPU before all measurements. We use the tool *stress-ng*[2] to maintain a CPU load of 95% for 5 seconds. In previous measurements with the same setup, we have seen that a longer warm-up phase does not provide any additional benefit.

6.2 External Validity

In this section, we discuss the generalizability of our results.

Within the constrained time frame available for a master's thesis, we were able to measure four case studies, each with around 20 *releases* and around 1 000 *configurations*.

With *infrastructure software* and *application software*, we considered two different types of software systems for our case studies. We measured *application software* from two different domains, the database servers *HSQLDB* and *PostgreSQL*, and the web server *Apache httpd*. With the video encoder *libvpx VP8* we only have a single *application software* case study from a single domain. To generalize our results for all types of software, we would have needed to measure more case studies from different domains. Nevertheless, since we have observed similarities between all of our case studies, we assume that many other software systems show a similar behaviour with regard to *performance* and *energy consumption*.

Across all case studies, we considered a wide range of different *configuration options*, making us confident that these are generally representative for *configuration options* used in practice – at least in software systems from the domains we covered with our case studies.

[1] For most *configurations* it is much higher than that, for many even more than 50%.

[2] http://kernel.ubuntu.com/~cking/stress-ng/ – last visited on 2019-09-14

7. Conclusion and Future Work

Contemporary software systems are complex and evolving. This evolution affects functionality and *non-functional properties* alike. As a basis for performance and energy optimizations in such complex software systems, we aimed at understanding the relation between these two *non-functional properties*, by investigating the evolution of *performance* and *energy consumption* in configurable software systems. More specifically, we explored changes between consecutive *release* in the *performance* and *energy consumption*, as well as changes in the *correlation* between *performance* and *energy consumption*. For this purpose, we measured four case studies – *HSQLDB*, *Apache httpd*, *PostgreSQL* and *libvpx VP8* – and evaluated the results to answer our research questions.

In the first research question, we compared *performance* and *energy consumption* between *releases* to find out if and how the *performance* and *energy consumption* evolves. Additionally, we used *performance-influence and energy-influence models* to attribute changes to specific *configuration options* and interactions. We found that *performance* and *energy consumption* change over time, with *substantial changes* in 27% of measured *releases* for the *performance* and in 26% of *releases* for the *energy consumption*. We also found that changes usually affect these two *non-functional properties* in the same way, with all of our observed *substantial changes* affecting the *performance* and 95% affecting the *energy consumption*. Changes are predominantly negative. In many cases we were able to attribute changes to specific *configuration options* or confirm that changes are independent from *configuration options* and affect all *configurations* alike. For our case studies, we could directly attribute 33% of observed changes to a specific *configuration option* or interaction.

In the second research question, we considered the *correlation* between *performance* and *energy consumption* and investigated whether this *correlation* changes over time. We found instances of changes in the *correlation* in 16% of measured *releases*. However, we were not able to draw clear conclusions from these observations, because the *correlation* values are heavily influenced by noise. In cases where we could clearly identify changes, these were often directly related to changes in the *performance* and *energy consumption*. A clear connection between changes in the *correlation* and specific *configuration options* was only visible in a single *release*.

Future Work

In our evaluation we discovered a limitation in the iterative learning approach of *SPL Conqueror*, which did not find influences for interactions between more than two *configuration options*. Future work could replace or complement this learning approach with other methods to determine the *performance-influences* and *energy-influences* of specific *configuration options*. By improving the quality and level of detail of the *influence models*, combined with a larger number of measured *releases*, one could more clearly and easily find relations between specific *non-functional property* value changes and specific *configuration options*.

In this work, we primarily used our measurement results to explore the evolution of *performance* and *energy consumption*. In future work, documentation – such as change logs or commit messages – or the source code could additionally be used to further pinpoint causes of changes and find reasons for specific behaviour of *performance*, *energy consumption* and their *correlation*.

A. Appendix

A.1 Content of the Accompanying CD

In this section, we list the files and directories that are included on the accompanying CD.

The file `thesis.pdf` is this thesis as PDF file.

The directory `casestudies` contains a subdirectory for each of the case studies, each containing the files used to prepare and run the case studies and a directory `results` which contains the measurement results. Some subdirectories containing a large number of small files have been archived as ZIP files. Large files, such as the source code repositories of the case studies and the 'Sintel trailer', which is used as workload for *libvpx VP8*, are not included due to the space constraints of a CD. These files can easily be obtained on the Internet from the websites linked in the footnotes of the section on case studies (Section 4.2).

The directory `plots` contains all plots from this thesis and additional plots that we used to analyse the results but did not include in the thesis. To view these plots, open the file `index.html` in a web browser. Additionally, the scripts used to generate these plots are included in a subdirectory `scripts`.

Lastly, the file `energymetering.zip` contains the 'EnergyMetering' measurement framework which we used to measure and analyse the case studies.

A.2 List of Releases

The following table lists all *releases* that we measured for the case studies.

HSQLDB	*Apache httpd*	*PostgreSQL*	*libvpx VP8*
2.1.0	2.2.0	8.3.0	v0.9.1
2.2.0	2.2.3	8.3.5	v0.9.2
2.2.1	2.2.6	8.4.0	v0.9.5
2.2.2	2.2.9	8.4.2	v0.9.6
2.2.3	2.2.11	9.0.0	v0.9.7
2.2.4	2.2.13	9.0.4	v0.9.7-p1
2.2.5	2.2.15	9.1.0	v1.0.0
2.2.6	2.2.17	9.1.3	v1.1.0
2.2.7	2.2.20	9.2.0	v1.2.0
2.2.8	2.2.22	9.2.4	v1.3.0
2.2.9	2.4.2	9.3.0	v1.4.0
2.3.0	2.4.4	9.3.4	v1.5.0
2.3.1	2.4.7	9.4.0	v1.6.1
2.3.2	2.4.10	9.4.4	v1.7.0
2.3.3	2.4.16	9.5.0	v1.8.0
2.3.4	2.4.18	9.5.3	
2.3.5	2.4.23	9.6.0	
2.4.0	2.4.27	9.6.3	
2.4.1	2.4.33	10.0	
	2.4.35	10.4	
	2.4.38	11.0	
		11.2	

A.3 Additional Plots

This section contains additional plots that we omitted from the previous chapters for brevity.

Figure A.1 was omitted from Section 5.1 (Evaluation of RQ1.1).

Figure A.2 was omitted from Section 5.3 (Evaluation of RQ1.3).

Figure A.3 was omitted from Section 5.5 (Evaluation of RQ2.2).

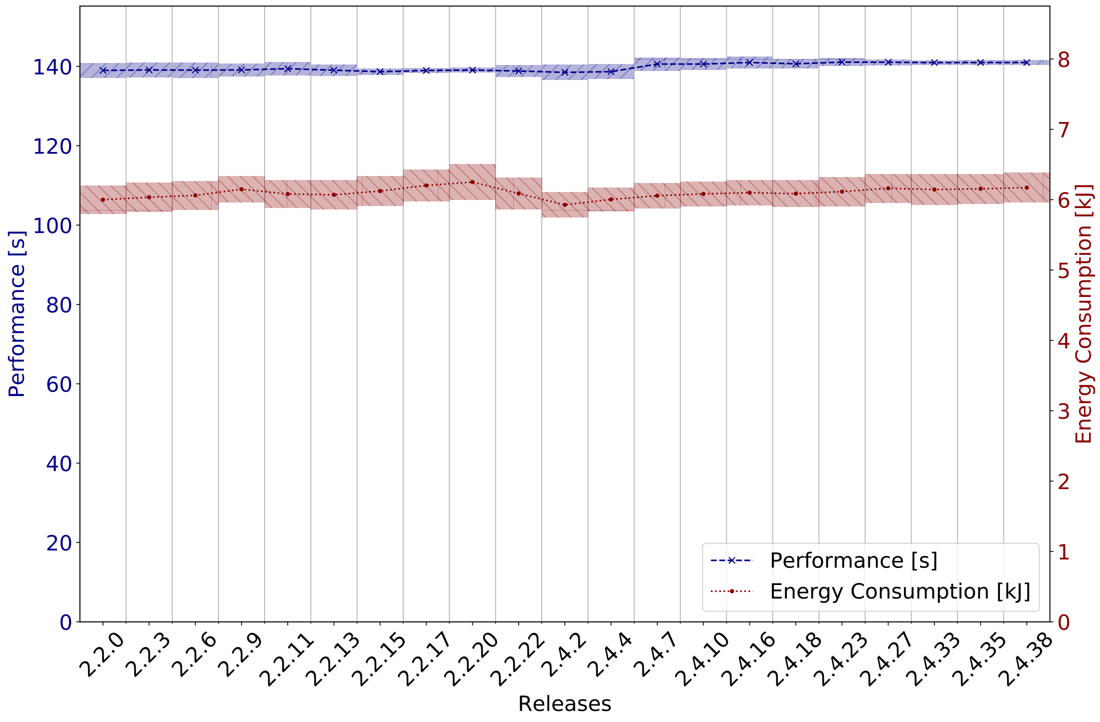


Figure A.1: Mean *performance* and *energy* of the *Apache httpd* case study. Boxes indicate threshold values. Percentages indicate relative changes for *substantial changes*.

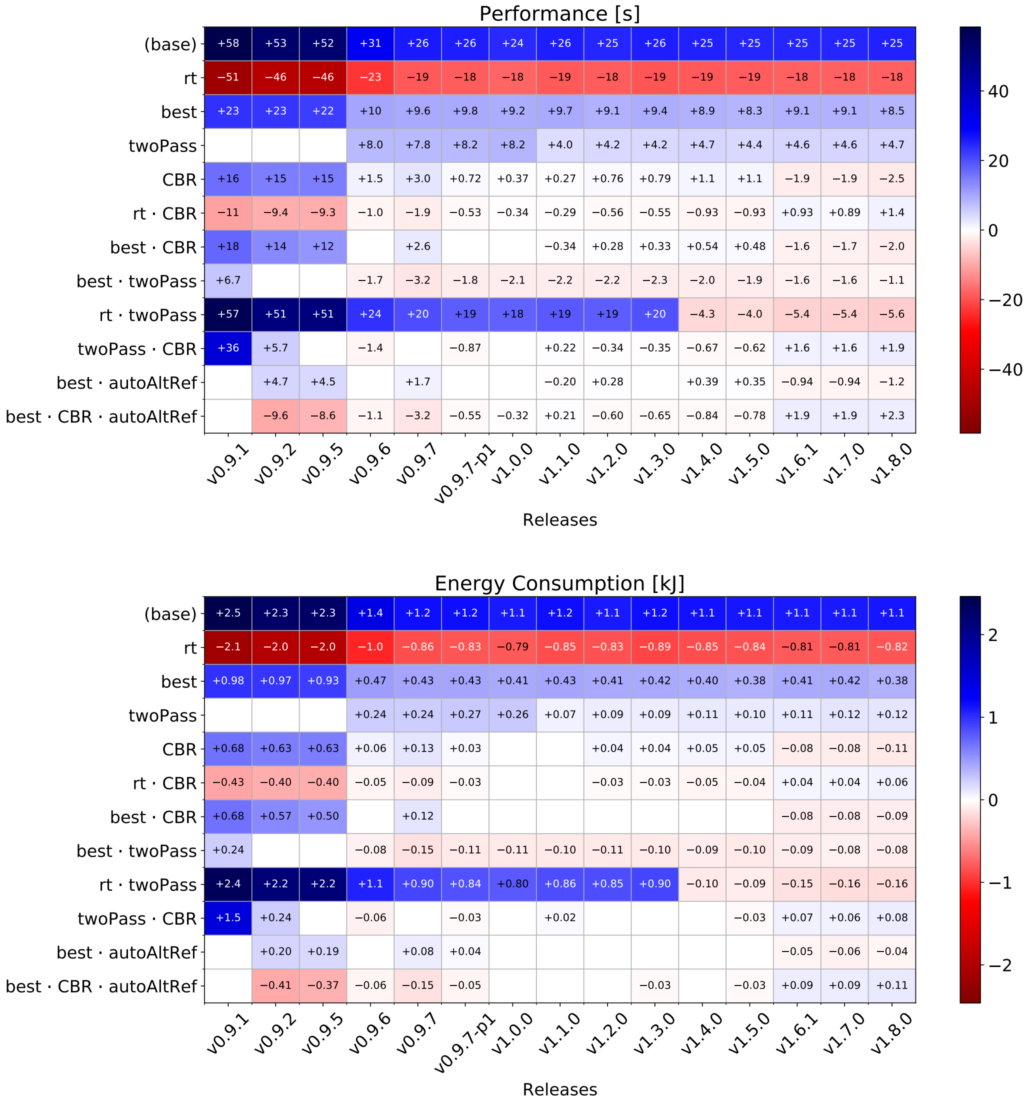


Figure A.2: *Performance-influence and energy-influence models* for the *libvpx VP8* case study. Only values exceeding the threshold are shown. The models only represent *configurations* with *threads* = 2. The option 'constantBitrate' is abbreviated as 'CBR' for brevity. The last influence was not found by iterative learning.

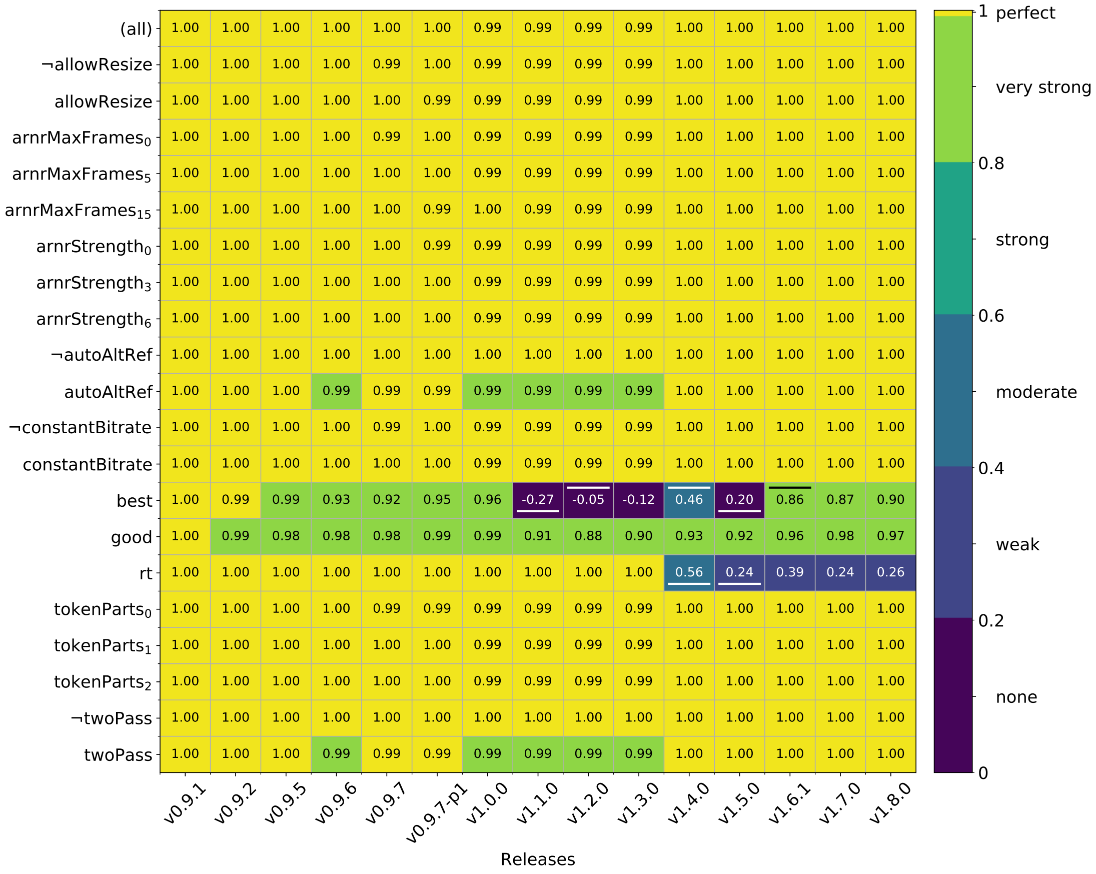


Figure A.3: *Pearson correlation coefficient* for subsets of the *configuration space* for each *release* of the *libvpx VP8* case study. A bar over a correlation value indicates a positive *substantial change* from the previous *release*, a bar under a value indicates a negative *substantial change*. Only *configurations* with *threads* = 2 are considered.

Bibliography

[BCHC09] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. *Pearson Correlation Coefficient*, pages 1–4. Springer Berlin Heidelberg, 2009. (cited on Page 15)

[CAKLR11] Lauro Beltrao Costa, Samer Al-Kiswany, Raquel Vigolvino Lopes, and Matei Ripeanu. Assessing Data Deduplication Trade-offs from an Energy and Performance Perspective. In *International Green Computing Conference and Workshops (IGSC)*. IEEE, 2011. (cited on Page 3)

[GBE07] Andy Georges, Dries Buytaert, and Lieven Eeckhout. Statistically Rigorous Java Performance Evaluation. *ACM SIGPLAN Notices*, 42(10):57–76, 2007. (cited on Page 30)

[Has19] Johannes Hasreiter. Evolution of Performance Influences in Configurable Systems. University of Passau, 2019. Master's thesis. (cited on Page 1, 5, and 27)

[JSV$^+$17] Pooyan Jamshidi, Norbert Siegmund, Miguel Velez, Christian Kästner, Akshay Patel, and Yuvraj Agarwal. Transfer Learning for Performance Modeling of Configurable Systems: An Exploratory Analysis. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 497–508. IEEE, 2017. (cited on Page 4)

[Leh80] Meir M. Lehman. Programs, Life Cycles, and Laws of Software Evolution. *Proceedings of the IEEE*, 68(9):1060–1076, 1980. (cited on Page 9)

[MAS19] Stefan Mühlbauer, Sven Apel, and Norbert Siegmund. Accurate Modeling of Performance Histories for Evolving Software Systems. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019. (cited on Page 5)

[MDHS09] Todd Mytkowicz, Amer Diwan, Matthias Hauswirth, and Peter F. Sweeney. Producing Wrong Data Without Doing Anything Obviously Wrong! *ACM SIGPLAN Notices*, 44(3):265–276, 2009. (cited on Page 73)

[Pea96] Karl Pearson. Mathematical Contributions to the Theory of Evolution. iii. Regression, Heredity, and Panmixia. *Philosophical Transactions of* *the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 187:253–318, 1896. (cited on Page 15)

[SBDD13] Juan Pablo Sandoval Alcocer, Alexandre Bergel, Stéphane Ducasse, and Marcus Denker. Performance Evolution Blueprint: Understanding the Impact of Software Evolution on Performance. In *IEEE Working Conference on Software Visualization (VISSOFT)*, pages 1–9. IEEE, 2013. (cited on Page 5)

[SGAK15] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kästner. Performance-influence Models for Highly Configurable Systems. In *Proceedings of the Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, pages 284–294. ACM, 2015. (cited on Page 1, 2, 4, 8, and 13)

[THS10] Dimitris Tsirogiannis, Stavros Harizopoulos, and Mehul A. Shah. Analyzing the Energy Efficiency of a Database Server. In *Proceedings of the International Conference on Management of Data (SIGMOD/PODS)*, pages 231–242. ACM, 2010. (cited on Page 1, 3, and 10)

[Wer17] Niklas Werner. Performance and Energy Interactions of Configurable Systems. University of Passau, 2017. Bachelor thesis. (cited on Page 1, 4, 7, and 72)

[XJF$^+$15] Tianyin Xu, Long Jin, Xuepeng Fan, Yuanyuan Zhou, Shankar Pasupathy, and Rukma Talwadker. Hey, You Have Given Me Too Many Knobs!: Understanding and Dealing with Over-designed Configuration in System Software. In *Proceedings of the Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, pages 307–319. ACM, 2015. (cited on Page 8)

[XTW10] Zichen Xu, Yi-Cheng Tu, and Xiaorui Wang. Exploring Power-performance Tradeoffs in Database Systems. In *IEEE International Conference on Data Engineering (ICDE)*, pages 485–496. IEEE, 2010. (cited on Page 4)

Eidesstattliche Erklärung:

Hiermit versichere ich an Eides statt, dass ich diese Masterarbeit selbständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe und dass alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind, sowie dass ich die Masterarbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Niklas Werner

Passau, den 26. September 2019