University of Passau

Department of Informatics and Mathematics

Bachelor Thesis

Performance and Energy Interactions of Configurable Systems

Author:

Niklas Werner

October 11, 2017

Advisors:

Prof. Dr.-Ing. Sven Apel

Chair of Software Engineering I

Christian Kaltenecker

Chair of Software Engineering I

Alexander Grebhahn

Chair of Software Engineering I

Werner, Niklas:
Performance and Energy Interactions of Configurable Systems
Bachelor Thesis, University of Passau, 2017.

Abstract

In the past, performance was the focus of optimizations in software, which is nowadays usually configurable. Recently, energy consumption became another important factor. Existing studies have compared performance and energy consumption with different results. Other studies have investigated the influence of different configuration options on the performance. In this thesis, we compare the performance and energy consumption for three case studies: HSQLDB, Apache, and x264. Additionally, we generate performance-influence and energy-influence models with SPL Conqueror and compare them. Before we conduct the case studies, we perform sanity checks for our measurement setup. We find, that performance and energy consumption correlate for application software, but not for infrastructure software. The influence models are almost equal for application software and different for infrastructure software.

Contents

List of Figures

List of Tables

1. Introduction

While, in the past, the focus of optimizations in software was on performance, recently also energy consumption becomes increasingly important. Additionally, most software systems are nowadays configurable to adapt to the requirements of different customers and fields of application.

Existing studies have focused on the influence of configuration options on the performance. Other studies have compared performance and energy consumption – with different results. Some studies come to the conclusion, that a correlation between performance and energy consumption exists, while others find that such a correlation does not exists.

In this thesis, we combine these two areas of interest and compare the performance and energy consumption for configurable software systems and investigate the influences from configurations options on both the performance and energy consumption. In an attempt to explain the different results regarding a correlation between performance and energy consumption in existing studies, we identify two different types of software systems – infrastructure software and application software – that exhibit a different behaviour regarding this correlation.

For this comparison of performance and energy consumption, we conduct three case studies from different domains: the database server *HSQLDB*, the web server *Apache*, and the video encoder *x264*.

Mytkowicz et al. [MDHS09] show, that there are many factors that can influence a measurement and possibly even lead to wrong results. For that reason, we perform sanity checks for our measurement setup before we conduct the case studies.

Using the results from the case studies, we investigate the correlation between performance and energy consumption and compare the influences from configuration options on both performance and energy consumption.

This thesis is structured as follows:

Following this introduction, we provide an overview on related work in Chapter 2, where we describe and compare three studies that compare performance and energy consumption and that come to different conclusions.

Then, we provide some background necessary to understand the topics covered in this thesis, and introduce relevant terms in Chapter 3. First, we describe the difference between infrastructure software and application software. Then, we explain the most important terms for this thesis: configurable system, performance, and energy consumption. Finally, we provide some information on performance-influence and energy-influence models as well as the Pearson correlation coefficient, which we use for the evaluation.

The general background is followed by the more specific methodology in Chapter 4. In that chapter, we introduce our research questions and the case studies we conduct to answer the research questions. We introduce each of the three case studies – *HSQLDB*, *Apache*, and *x264* – and describe the configuration options we used.

In Chapter 5, we describe the general experimental setup that is common for all case studies, and specific additions for each case study. Here, we also conduct sanity checks for the measurement setup.

Afterwards, we use the results from the case studies for the evaluation in Chapter 6. We answer each research question in one section by describing the operationalization and presenting and discussing the results.

Every experimental setup contains potential threats to the validity of the results. In Chapter 7, we discuss both the internal and external validity of our findings.

Finally, in Chapter 8, we conclude the thesis with a summary and an outlook on possible future work.

2. Related Work

Several papers investigate the performance and energy consumption of different types of systems. However, they differ in their measurement setup and, consequently, they do not all come to the same conclusion.

Tsirogiannis et al. [THS10] analyse the energy efficiency of database systems. This is accomplished by investigating the performance and energy consumption of specially developed 'micro-benchmarks' which implement core database operations. Additionally, the energy efficiency of specific algorithms in *PostgreSQL* is evaluated. The energy consumption is measured for the individual hardware components in their experimental setup and the evaluation focuses on the energy consumption of the CPU. They compare the performance, which they define as the ratio of work done to run time, with the energy efficiency, which is the ratio of work done to energy consumption. For various combinations of hardware and software features, they compare the configurations with the highest performance and the highest energy efficiency. Their results show that the most energy efficient configuration usually has the best performance.

Besides, Costa et al. [CAKLR11] evaluate power–performance tradeoffs in data deduplication systems. Unlike in the study by Tsirogiannis et al., Costa et al. define the performance as the ratio of run time to work done and they define the energy efficiency as the ratio of energy consumption to work done. The energy consumption is measured for the whole system and not for individual components like in the previous study. For varying similarity ratio in the test data, they determine the break-even points between enabled and disabled data deduplication. They do this independently for performance and energy efficiency and compare the break-even points of both. They find that the existence of such tradeoffs depends on the used hardware because new hardware systems are increasingly power-proportional, i.e., the power consumption is proportional to the resource utilization. For such new systems, tradeoffs exist between power and performance optimizations.

Another evaluation of performance and power consumption of database systems is conducted by Xu et al. [XTW10]. Their goal is to explore power–performance tradeoffs. For the measurements, *PostgreSQL* is used with benchmark scenarios derived

from the TPC benchmarks. The energy consumption is measured for the whole system like in the study by Costa et al., but unlike in the other two studies, they additionally measure the power consumption and have no notion of energy efficiency. Likewise, instead of calculating a value for the 'performance', they directly compare the run time. Comparing the power and energy consumption and the run time for different workloads, they conclude that power–performance tradeoffs exist and that software which is aware of power consumption, could apply optimizations that greatly decrease the power consumption while only minimally affecting performance.

While all three of those studies investigate the performance and energy or power consumption, they come to different conclusions. The findings of Tsirogiannis et al. [THS10] indicate that energy consumption correlates with performance. Both the studies of Xu et al. [XTW10] and Costa et al. [CAKLR11] do not indicate such a strong correlation. This indicates, that the correlation between performance and energy consumption is not the same for all types of software systems. Software that runs continuously whether or not it is actively used, seems to have a different behaviour regarding performance and energy consumption than software that only runs for a limited time during which it actively used.

3. Background

In this chapter, we provide an overview of the general topic of this thesis – performance and energy measurements – and explain the terminology used throughout the following chapters.

First, in Section 3.1, we describe two different types of applications. Since this thesis investigates performance and energy interactions of configurable systems, we then provide an introduction to the integral terms for this topic. We provide an introduction to the term 'configurable system' and related terms in Section 3.2. Then we introduce the term 'performance' and how it is defined in this thesis in Section 3.3 and the term 'energy consumption' and how it can be measured in Section 3.4. In Section 3.5, we provide an overview on performance-influence and energy-influence models as a way to compare performance and energy consumption. Finally, in Section 3.6, we introduce the Pearson correlation coefficient which we will use to investigate linear correlations in our results.

3.1 Application Types

The related work in Chapter 2 indicates that performance and energy interactions are different depending on the type of software system. In this thesis, we distinguish between two different types of systems: server-like software that runs continuously ('24/7') and software that runs only for a limited amount of time to perform a certain task.

We denote software that runs continuously as *infrastructure software*. Examples for this type of software include web servers or database servers. *Infrastructure software* runs even when it is not actively used, and consequently, a computer has to run permanently and consumes energy permanently.

Software that only runs when needed will be denoted as *application software*. Examples for this type of software include video encoding or compression tools. In contrast to *infrastructure software*, *application software* only runs when it is used. When it is not used, it does not run. Thus, no computer has to run and consume energy while the application is not used.

3.2 Configurable Systems

A configurable (software) system is software with configuration options. Configuration options are features. A feature is a distinct piece of functionality, which can be optional or mandatory. Optional features can be either selected or deselected. They can be 'turned on and off'. There are two different types of such optional features, distinguished by when they can be turned on and off. Variability at compile time is achieved with features that are optional at compile time but cannot be changed after compilation. On the other hand, variability at runtime is achieved with features that are still toggleable after compilation, for example, by passing command line arguments to the software.

Throughout this thesis, the term *feature* will be used to denote a usually optional feature, i.e. a configuration option.

Usually, not all combinations of *features* are valid. For example, a *feature* might actually be required. Or two *features* might be mutually exclusive. The term *configuration* is used to denote a valid combination of *features*.

Features and *configurations* can be represented mathematically as sets. We denote the set of all *features* (configuration options) as $\mathcal{O}$ and the set of all *configurations* as C. A *configuration* $c \in C$ is a function $\mathcal{O} \to \{0, 1\}$. For a *feature* $o \in \mathcal{O}$, $c(o) = 0$ if the *feature* is not selected for the *configuration* c and $c(o) = 1$ if the *feature* is selected.

A *feature model* consists of a set of features and their constraints. Consequently, a *feature model* describes all valid configurations.

A *feature diagram* is a graphical notation of a *feature model* in the form of a tree. Figure 3.1 shows an example for a *feature diagram*.

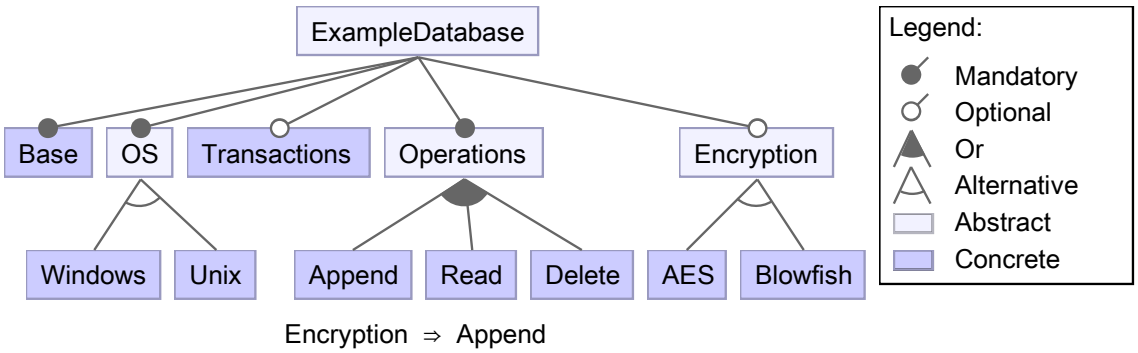

Figure 3.1: Example for a *feature diagram* showing the *features* and their constraints for the system 'ExampleDatabase'

In this example, the software system 'ExampleDatabase' has the *concrete features* 'Base', 'Windows', 'Unix', 'Transactions', 'Append', 'Read', 'Delete', 'AES', and 'Blowfish'. 'OS', 'Operations', and 'Encryption' (and 'ExampleDatabase') are not actually *features* in the sense that they provide functionality. Rather they are *abstract features*, used in the diagram to group other features, which allows expressing certain constraints between features. Different types of edges are used to represent different types of constraints. The features 'Base', 'OS', and 'Operations' are

mandatory, as denoted by an edge with a filled circle. They are selected for all *configurations*. The features 'Transactions' and 'Encryption' are optional, as denoted by the edges with a blank circle. The blank arc between the edges of 'Windows' and 'Unix' denote an *alternative*. Exactly one of these features is required. This constraint can be seen as an *exclusive or*. On the other hand, edges with a filled arc denote an *inclusive or* constraint. At least one of the features 'Append', 'Read', and 'Delete' is required. A feature that appears below another feature in a *feature diagram* must not be used when the feature above it is not used. In the example, the features 'AES' or 'Blowfish' can only be used for *configurations* with the feature 'Encryption'.

While constraints between features can be arbitrarily complex in a *feature model*, the tree representation of a *feature diagram* can only express a few basic constraints as described in the example above. To ensure that the *feature diagram* accurately describes the *feature model* with all its constraints, additional constraints that cannot be expressed with the tree, can be added to the *feature diagram* as Boolean expressions. These expressions contain the features as variables and have to be true for valid *configurations*. Such additional constraints that complement the tree in a *feature diagram* are called *cross-tree constraints*.

In the example, ***Encryption*** $\Rightarrow$ ***Append*** is such a constraint that cannot be represented in the tree. It specifies, that for *configurations* with 'Encryption', the feature 'Append' is required.

Usually, a mandatory feature is included in the feature model that does not represent a configuration option but rather the core functionality of the system, that is available regardless of any configuration options. In the above example and throughout this thesis, we call this mandatory feature 'Base'.

The term *feature* as defined above, only refers to the *concrete features* of a *feature diagram* since the *abstract features* of a *feature diagram* do not correspond to functionality.

3.3 Performance

There are different ways to measure what can be called the performance of a system. In fact, how performance can be measured depends on one's definition of the term. For example, performance can be defined as the time taken to produce a result or as some quality attribute of the result.

Tsirogiannis et al. [THS10] define performance as the ratio of work done to time:

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

In this thesis, we always determine performance by the time taken to produce a result, or more specifically, the time taken to execute a benchmark. We do not consider any quality attributes of the result. Thus, the work done is always the same (the execution of a specific benchmark). We define that constant work to be 1. The performance is then only defined by the time taken to execute a benchmark:

$$Performance = \frac{1}{Run\ time} \tag{3.2}$$

Since, in this thesis, time is the only factor influencing performance, we use a slightly different and simpler definition of performance throughout this thesis. We define *performance* as follows:

$$Performance = Run\ time \tag{3.3}$$

The difference between *infrastructure software* and *application software* has an impact on how performance can be measured.

Application software runs on a single computer and the measured software directly executes the benchmark. Consequently, the *performance* is determined by the execution time of the measured software.

Infrastructure software, on the other hand, runs in a client–server setup where the measured software runs as a server and the benchmark runs as a client application. Hence, while the *performance* is still determined by the execution time of the benchmark, it is not the execution time of the measured *infrastructure software*, but rather the execution time of the client application.

Note, however, that in this context, execution time does not necessarily mean that the actual execution time from the start of a process to the end is measured, but it may be possible to extract a more accurate value from the output of the process. For example, a benchmark application might write accurate start and end times of the benchmark to a log file. These times can more accurately represent the run time of the benchmark and we use them whenever they are available.

3.4 Energy Consumption

The energy consumption is another integral term for this thesis.

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:

$$E = \sum_{t=\frac{t_{start}}{\Delta t}}^{\frac{t_{end}}{\Delta t}} P(t\ \Delta t) * \Delta t \tag{3.4}$$

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 period with which power consumption values can be measured.

This calculation is simplified by the fact that with our experimental setup we can set $\Delta t = 1s$. In other words, the *energy consumption* is calculated as the sum of power consumption values for each second during the measurement:

$$E = \sum_{t=t_{start}}^{t_{end}} P(t) * 1s \tag{3.5}$$

t_{start} and t_{end} are measured in seconds.

The difference between *infrastructure software* and *application software* affects not only the definition of *performance* but also the definition of *energy consumption*. While the notion of *energy consumption* for *application software* is quite straightforward, the same is not the case for *infrastructure software*. For *application software*, which only runs when needed, the *energy consumption* is measured during the whole time the application runs. On the other hand, for *infrastructure software*, the software runs also when it is not actively used. In fact, it runs continuously – idly while not used – so 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.

In this thesis, we answer the question by running all the measurements of each case study for the same fixed duration, which includes the energy consumption during active use and the idle energy consumption until the end of the measurement. This simulates a real world scenario where the *infrastructure software* is used periodically with a fixed period. Only this combination of active use and idle period define the *energy consumption* for *infrastructure software* for this thesis.

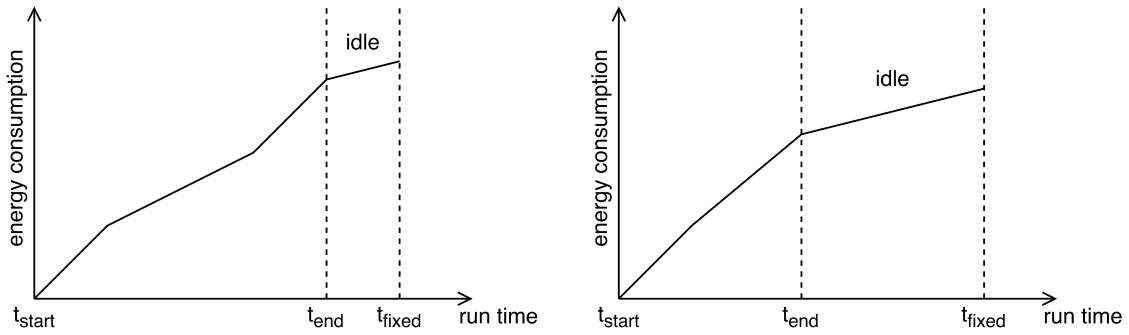

Figure 3.2: Fixed duration for the *energy consumption* of *infrastructure software*. The left graph shows the *energy consumption* over the course of a measurement for a slow *configuration* and the right graph for a fast *configuration*. The *energy consumption* after the end of the benchmark (t_{end}) until the end of the measurement with a fixed duration (t_{fixed}) is the idle energy consumption of the system.

The fixed duration for the *energy consumption* of *infrastructure software* is illustrated in Figure 3.2. The measurement and benchmark start at t_{start} and the benchmark ends at t_{end}. The measurements continues until t_{fixed}, which is the same for all *configurations*. In the experimental environment, t_{fixed} is the end of the measurement. In a real world scenario, it would be the start of the next period of active use. In the figure, there is a slow configuration on the left and a fast configuration on the right. Both measurements include the idle energy consumption until t_{fixed}, which is more for the fast configuration.

3.5 Performance-Influence and Energy-Influence Models

A performance-influence model describes the performance of *configurations*. More specifically, it describes the influence of individual *features* and interactions between *features* on the performance of all *configurations* ([SGAK15]).

Interactions of *features* are influences that only appear with a certain combination of *features*. They can simply be caused by code that is only executed for a combination of *features*, but interactions can also be less obvious. For example, in a database system, the code of a *feature* 'Encryption' might not change depending on whether the *feature* 'Compression' is enabled or not, however, at runtime the *feature* 'Compression' might still have an influence on the performance of 'Encryption', because 'Compression' will result in 'Encryption' having to encrypt a smaller amount of data.

These influence models are not restricted to describing the performance of a system, but can be used for other measurable attributes as well, for example, *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*.

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

$$\Pi(c) = \beta_0 + \sum_{i \in \mathcal{O}} \phi_i(c(i)) + \sum_{i..j \in \mathcal{O}} \Phi_{i..j}(c(i)..c(j)) \tag{3.6}$$

$\Pi(c)$ is the *performance* of a *configuration* $c \in C$. The set of all *features* $\mathcal{O}$ and the set of all *configurations* C are defined in Section 3.2. β_0 is a constant base value for the *performance* of all *configurations*, which is independent from the *feature* selection. $\phi_i(c(i))$ is the influence on the *performance* of a single *feature* i based on its presence in the *configuration* c and $\Phi_{i..j}(c(i)..c(j))$ is the influence on the *performance* from multiple *features* $i..j$ based on their presence in the *configuration* c.

We denote a *performance-influence model* as $\Pi(c)$ and an *energy-influence model* as $E(c)$.

For the example database system from Figure 3.1 in Section 3.2, the following might be an excerpt from the *performance-influence model* for the features 'Transactions' (T), 'Append' (A), 'Read' (R), 'Delete' (D), and 'Blowfish' (B):

$$\begin{aligned} \Pi(c) = 100 &+ 25 \cdot c(T) + 20 \cdot c(B) + 10 \cdot c(T) \cdot c(D) \\ &+ 35 \cdot c(A) \cdot (B) + 5 \cdot c(T) \cdot c(A) \cdot c(B) \end{aligned} \tag{3.7}$$

In the example, 'Transactions' and 'Delete' interact, as do 'Append' and 'Blowfish'. Additionally, there is an interaction between the three *features* 'Transactions', 'Append', and 'Blowfish'. From the model, the *performance* of a configuration could be calculated by simply plugging in a *configuration* c into the formula.

In addition to introducing *performance-influence models*, Siegmund et al. [SGAK15] also propose a machine-learning algorithm to derive such models from a sample of measured *configurations* and provide an implementation of the algorithm with the tool *SPL Conqueror*[1].

We use *SPL Conqueror* to generate the *performance-influence models* and *energy-influence models* for this thesis.

Our primary use for these models is to generate both *performance-influence models* and *energy-influence models* for each case study and compare the models to analyse differences between *performance* and *energy consumption*.

3.6 Correlation

To compare *performance* and *energy consumption*, we will investigate the correlation between them. For this purpose we calculate a correlation coefficient.

The *Pearson correlation coefficient* was developed by Pearson towards the end of the 19th century ([Pea96]). It is a measurement for the linear correlation between two variables. Benesty et al. [BCHC09] define the *Pearson correlation coefficient* ρ as follows:

$$\begin{aligned} \rho(a,b) &= \frac{E(a,b)}{\sigma_a \sigma_b} \\ E(a,b) &= \frac{1}{n}\sum_{i=1}^{n}(a_i - \bar{a})(b_i - \bar{b}) \\ \sigma_a &= \frac{1}{n}\sum_{i=1}^{n}(a_i - \bar{a})^2 \end{aligned} \tag{3.8}$$

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

The *Pearson correlation coefficient* can range 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. Absolute values less than 0.5 are usually said to indicate only weak correlation and absolute values greater than 0.8 indicate strong correlation.

We will use the *Pearson correlation coefficient* to investigate the correlation between *performance* and *energy consumption*.

[1] http://www.infosun.fim.uni-passau.de/se/projects/splconqueror/ – last visited on 2017-09-05

4. Methodology

In this chapter, we introduce the general methodology that we use in this thesis. The methodology comprises of the research questions and the case studies we use to answer them. We develop the research questions in Section 4.1 and introduce the case studies in Section 4.2.

4.1 Research Questions

In this section, we summarize the previous chapters to develop our research questions and introduce the case studies we conduct to answer the research questions.

Our first research question will cover the general relationship between *performance* and *energy consumption* by examining whether a correlation exists between them. Since the related work in Chapter 2 indicates that *infrastructure software* and *application software* behave differently regarding *performance* and *energy consumption*, we will also examine the correlation independently for both types of software systems.

In our second research question, we will further analyse the differences between *performance* and *energy consumption* by comparing the *performance-influence models* and *energy-influence models* obtained with *SPL Conqueror* for our case studies.

Overall, this leads us to the following list of research questions:

RQ1: Is there a correlation between *performance* and *energy consumption*?

RQ1.1: Is there a correlation between *performance* and *energy consumption* in *infrastructure software*?

RQ1.2: Is there a correlation between *performance* and *energy consumption* in *application software*?

RQ2: Do the *performance-influence models* and *energy-influence models* contain the same terms?

4.2 Case Studies

To answer the research questions, we conducted three different case studies. To keep within reasonable bounds for a bachelor thesis, we selected *HSQLDB* and *Apache* as two examples for *infrastructure software* and *x264* as one example for *application software*.

In the following sections, we provide an overview of these three software systems and describe the *features* that we used.

4.2.1 HSQLDB

HyperSQL DataBase[1], or *HSQLDB*, is an open source relational database system written in Java. While *HSQLDB* can run as embedded database in applications, in which case it might behave like *application software*, we used it in standalone server mode, which makes it an example for *infrastructure software*.

Features

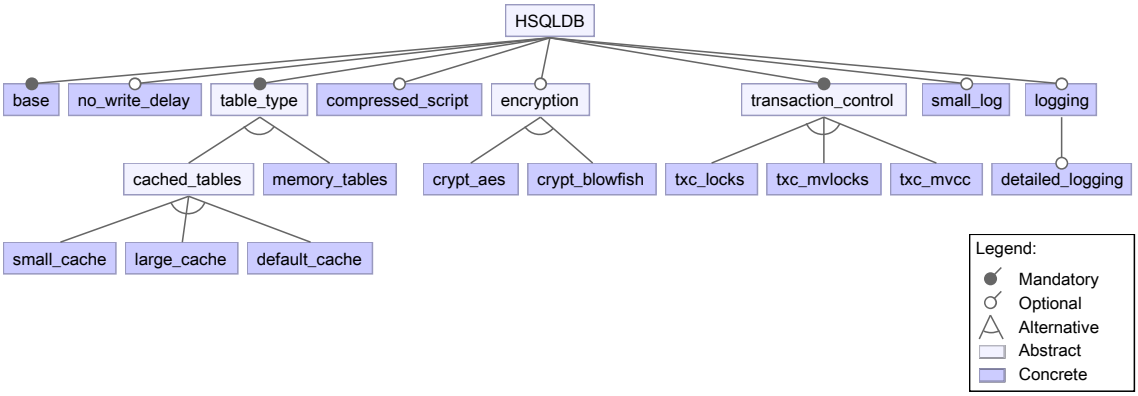

Figure 4.1: *Feature diagram* for the *HSQLDB* case study

In the following list, we describe the *features*[2] we used for the case study and in Figure 4.1 we show the corresponding *feature diagram*. The *feature model* for the *HSQLDB* case study describes 864 *configurations*.

no_write_delay: The write delay specifies the interval, in which changes to the database are persisted to the disk. The default value is 0.5 seconds. With the *feature* 'no_write_delay' enabled, the write delay is set to 0, which means changes are immediately written to disk.

small_cache, large_cache, default_cache: The default in *HSQLDB* is to keep tables completely in memory for faster read access. The three 'cache' features change the default table type to 'cached tables' which are only partially kept in memory. The default amount of memory used for cached tables is 10 MB. The *features* 'small_cache' and 'large_cache' change this cache size to 1 MB and 100 MB, respectively.

[1] http://hsqldb.org/ – last visited on 2017-08-26
[2] http://hsqldb.org/doc/2.0/guide/guide.pdf – last visited on 2017-09-13

memory_tables: This *feature* uses the default table type which completely caches tables in memory.

compressed_script: The 'script file' is used to persist a database on the disk. The default format is plain text. With the *feature* 'compressed_script', the script file is compressed.

crypt_aes, crypt_blowfish: With either of these *features* selected, the database contents that are written to the disk, are encrypted. AES encryption is used with 'crypt_aes' and Blowfish encryption is used with 'crypt_blowfish'.

txc_locks, txc_mvlocks, txc_mvcc: The three 'txc' features specify the concurrency control model (transaction control). The default 'two phase locking' is used with 'txc_locks', 'two phase locking with snapshot isolation' is used with 'txc_mvlocks', and 'txc_mvcc' uses the 'MVCC' model.

small_log: *HSQLDB* writes changes to a so called log file. When the log file reaches a certain size, the changes are written to the script file and the log file is cleared. By default, this happens when the log file reaches a size of 50 MB. With the *feature* 'small_log' selected, this limit is reduced to 5 MB.

logging, detailed_logging: The *feature* 'logging' enables logging events to a file with the default log level. The *feature* 'detailed_logging' increases the log level to include more details in the log file.

4.2.2 Apache

The *Apache HTTP Server*[3], or 'httpd', in this thesis simply *Apache*, is an open source HTTP server written in C. It is another example for *infrastructure software*.

Features

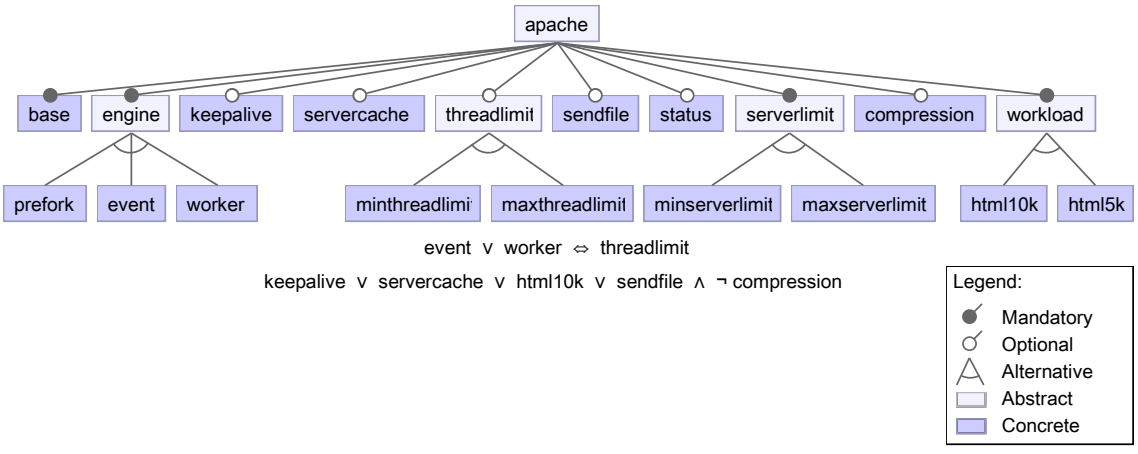

Figure 4.2: *Feature diagram* for the *Apache* case study

[3] http://httpd.apache.org/ – last visited on 2017-09-07

In Figure 4.2, we illustrate the *features* of this case study and their constraints. We describe the *cross-tree constraints* after the following description of the *features*[4]. The *feature model* describes 580 *configurations* for the *Apache* case study.

prefork, event, worker: Apache uses a 'Multi-Processing Module' (MPM) which is responsible for accepting and handling requests and creating child processes or threads for this purpose. The *features* 'prefork', 'event', and 'worker' select the MPM with the respective name.

keepalive: This *feature* allows *Apache* to use persistent HTTP connections that can be used for multiple requests. The same connection can be used for up to 65 requests. A connection is closed when it remains unused for 5 seconds.

servercache: Allows the server to cache responses in a shared memory-based cache.

minthreadlimit, maxthreadlimit: The 'ThreadLimit' value restricts the maximum number of threads per child process. This value can only be used with the 'event' and 'worker' MPMs. The *features* 'minthreadlimit' and 'maxthreadlimit' set this value to 64 and 128, respectively.

sendfile: The *feature* 'sendfile' allows *Apache* to use the operating systems capabilities to directly send a static file to the client without the need to buffer the file contents between reading and sending.

status: The *feature* 'status' enables the 'ExtendedStatus' of *Apache*, which keeps track of extended status information for requests and creates a utilization summary.

minserverlimit, maxserverlimit: The 'ServerLimit' together with the 'ThreadLimit' restricts the maximum number request workers. For both the MPMs 'event' and 'worker', the *features* 'minserverlimit' and 'maxserverlimit' set this value to 16 and 32, respectively. For the MPM 'prefork' (which does not support 'ThreadLimit'), the value is set to 256 and 512, respectively, by these features.

compression: Enables gzip compression for response bodies.

html10k, html5k: These *features* do not represent functionality of *Apache* but different workloads used for the case study. We used two HTML files of size 10 KB and 5 KB which are selected by the *features* 'html10k' and 'html5k', respectively.

Two *cross-tree constraints* were necessary for the *feature model* of the *Apache* case study.

The first constraint, $\boldsymbol{event} \vee \boldsymbol{worker} \iff \boldsymbol{threadlimit}$, models the restriction of the 'prefork' MPM that does not support setting the 'ThreadLimit'. Consequently, the *feature* 'threadlimit' can be used only with 'event' or 'worker'.

[4] https://httpd.apache.org/docs/2.4/en/ – last visited on 2017-09-13

The second constraint, $\boldsymbol{keepalive} \vee \boldsymbol{servercache} \vee \boldsymbol{html10k} \vee \boldsymbol{sendfile} \wedge \neg\boldsymbol{compression}$, is necessary due to a restriction in our experimental setup which is not capable of measuring certain *configurations* (see Section 5.3.2). It excludes combinations of *features* with the small html file ('html5k') but without the *features* 'servercache' and 'keepalive' when the *feature* 'compression' is selected and 'sendfile' is not selected, from the set of valid *configurations*. Note, that the logical and operator ($\wedge$) has a higher precedence than the logical or operator ($\vee$).

4.2.3 x264

x264[5] is an encoder for the video compression format *H.264*. It is an example for *application software*.

Features

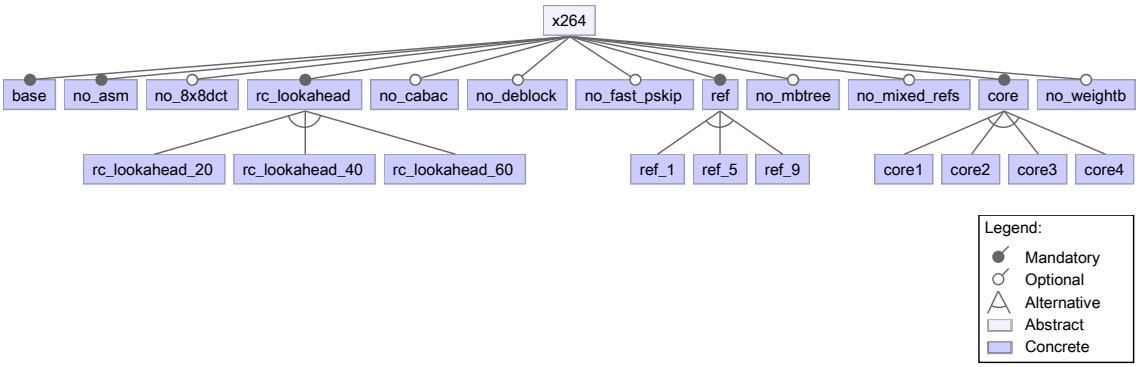

Figure 4.3: *Feature diagram* for the *x264* case study

Figure 4.3 shows a *feature diagram* for the *features* that we used for the case study. Most of the *features* correspond directly to command line parameters of *x264*. A short description of those *features* follows. More information about the *features* is available in the built-in documentation of *x264* (command line parameter `--fullhelp`). The *feature model* for the *x264* case study describes 4,608 *configurations*.

no_asm: Disables CPU optimizations, which may lead to unpredictable influences and are enabled by default.

no_8x8dct: Disables the 8x8 discrete cosine transform.

rc_lookahead_20, rc_lookahead_40, rc_lookahead_60: Changes the number of frames for frametype lookahead. The default value is 40. It can be changed to 20 and 60 with the *features* 'rc_lookahead_20' and 'rc_lookahead_60', respectively.

no_cabac: Disables context adaptive binary arithmetic coding (CABAC), which can decrease the output size but also decreases the decoding speed.

[5] https://www.videolan.org/developers/x264.html – last visited on 2017-08-26

no_deblock: Disables the deblocking loop filter which can increase the output quality but decrease both the encoding and decoding speed.

no_fast_pskip: Disables early skip detection for P-frames.

ref_1, ref_5, ref_9: Changes the number of reference frames. Higher values decrease the encoding speed and can increase the output quality. The *features* 'ref_1', 'ref_5', and 'ref_9' change this value to 1, 5, and 9, respectively.

no_mbtree: Disables macroblock-tree ratecontrol. Using 'no_mbtree' increases the encoding speed but decreases the output quality.

no_mixed_refs: Disables the use of different reference frames for each partition. This can increase the encoding speed.

core1, core2, core3, core4: The 'core' *features* do not change the functionality of *x264* but restrict the number of CPU cores that *x264* runs on. With the *features* 'core1', 'core2', or 'core3' selected, the *x264* process is restricted to use only the first 1, 2, or 3 cores of the CPU, respectively. The *features* 'core4' does not restrict the CPU cores available to *x264*.

no_weightb: Disables an optimization for B-frames that can increase the output quality.

5. Experimental Setup

In this chapter, we describe the general experimental setup, that we used for all case studies, and details that are specific to the experimental setup of the individual case studies.

5.1 Hardware

The case studies were conducted using workstation PCs from a cluster called *Zmiy*. The cluster *Zmiy* consists of 4 nodes with Intel Core i7 processors and 14 nodes with Intel Core i5 processors. In the following, these nodes are denoted as *i7 nodes* and *i5 nodes*, respectively. The *i7 nodes* have *Intel Core i7-4790* processors with 4 cores and 8 threads and a base frequency of 3.6 GHz. The *i5 nodes* have *Intel Core i5-4590* processors with 4 cores (and 4 threads) and a base frequency of 3.3 GHz. Both the *i7 nodes* and *i5 nodes* have 16 GB of RAM. For storage, the *i7 nodes* have HDDs and the *i5 nodes* have SSDs.

Each node has a 1 Gb/s *Intel Ethernet Connection I217-LM* network interface card and they are all connected to a single *Superstack 4 5500G* switch with 48 ports.

The cluster nodes use a minimal installation of Ubuntu 16.04.3 LTS.

Depending on the requirements of each case study, one or more nodes can be used for each configuration.

Since there are only 4 *i7 nodes* available, measuring all *configurations* on the *i7 nodes* would take much longer than using the *i5 nodes*, of which there are 14 available. Thus, only the *i5 nodes* were used for measuring the case studies.

5.2 Measurement Setup

All nodes are connected to power distribution units (PDUs) which measure the voltage, current, power consumption, power factor, and frequency, independently for all nodes. Note, that the PDUs do not measure the energy consumption, so the

energy consumption has to be calculated from the power consumption as explained in Section 3.4. All *i7 nodes* and 12 of the *i5 nodes* are connected to an *IPT iPower P1* PDU with 18 sockets and the remaining 2 *i5 nodes* are connected to another *IPT iPower P1* PDU with 3 sockets.

A small Java-based tool called *PDUGetterV2* was developed to periodically query the measured values from the PDUs and write them to a log file. The values are queried every 990 milliseconds to ensure that at least one set of values is logged every second. Values much lower than 1 second can result in the PDU not replying to queries and setting exactly 1 second as interval can lead to seconds without a value because querying the values from the PDUs and processing the response can take some time. The value of 990 milliseconds has proven to not lead to either of those problems.

In addition to logging the values measured by the PDUs, the CPU load for all used nodes is logged every second.

5.3 Case Studies

In addition to the general experimental setup from the previous sections, some parts of the experimental setup are specific to the individual case studies. We describe these specific additions in this section.

5.3.1 HSQLDB

The *HSQLDB* case study uses a client–server setup where the database system runs in server mode and the benchmark application acts as a client. Because only the *energy consumption* for the database system should be measured, the client runs on a separate node. We used version 2.4.0 of *HSQLDB*, running on the OpenJDK 1.8.0_131 64-Bit Server VM.

To generate load on the *HSQLDB* server, we used the *PolePosition*[1] benchmark. *PolePosition* is an open source benchmark for databases, which has built-in support for *HSQLDB*. *PolePosition* provides different scenarios ranging from read-focused to write-focused and from simple queries to very complex queries. We used version 0.6.0 of the *PolePosition* benchmark with a custom configuration using the three scenarios 'Complex', 'InheritanceHierarchy', and 'FlatObject', which consist of both reading and writing queries with varying complexity.

Since *HSQLDB* is written in Java and runs in the Java Virtual Machine (JVM), just-in-time compilation (JIT compilation) can substantially impact the start-up performance as sated by Georges et al. [GBE07]. We do not want to measure the start-up performance, so we execute a warm-up phase for the *HSQLDB* case study. This warm-up phase is executed before the measured benchmark run, using the same *HSQLDB* server instance. To make sure, that all server code that is used during the measurement is JIT compiled, the warm-up phase uses the *PolePosition* benchmark with a similar configuration as the one used for the measurement, but with fewer iterations.

[1] http://polepos.sourceforge.net/ – last visited on 2017-09-07

5.3.2 Apache

Since *Apache* is a web server, the *Apache* case study uses a client–server setup. *Apache* acts as server and the benchmark application *Gatling* acts as client. Just like in the *HSQLDB* case study, only the *energy consumption* of the server should be measured, so the client runs on a separate node.

We used version 2.4.25 of *Apache*. It was compiled with *APR* in version 1.5.2 and *APR-Util* in version 1.5.4.

We used *Gatling*[2] to benchmark *Apache*. *Gatling* is a tool for performance and load testing of web applications. We used *Gatling* version 2.2.5 to simulate 1000 clients (threads) each sending 1500 requests to *Apache* for static a HTML file. The size of the HTML file is 10 KB or 5 KB, depending on the used *configuration*. Since *Apache* cannot respond to that many requests within the default time-outs of *Gatling*, we increased the time-out values.

Even though *Gatling* is written in Scala – which means it is executed by the JVM – start-up performance due to JIT compilation is not an issue because we enabled the built-in warm-up phase included in *Gatling*.

Several restrictions limit the client's ability to maintain multiple open connections to the server. While we were able to increase the number of open file descriptors available to the client process, the number of available ports for outgoing connections was too low for some configurations where the server took particularly long to complete requests but was able to accept many concurrent connections. This is the case for *configurations* with the small html file ('html5k') that do not use the *features* 'keepalive' and 'servercache'. However, this problem does not affect *configurations* with 'sendfile', unless 'compression' is enabled as well. Since this restriction means that we cannot measure these *configurations*, we introduced it as a constraint in our *feature model* as shown in Figure 4.2 in Section 4.2.2.

Using multiple clients for this case study would likely have prevented the need for this constraint, but we observed considerably higher deviations between repeated measurements of the same *configuration* when using a second client.

5.3.3 x264

We used version 0.152.2851 (ba24899) of *x264*.

Since *x264* is *application software*, it directly executes the benchmark and no additional benchmark application is required. For a video encoder, the benchmark is a certain input video file. We used a lossless version of the 'Sintel' trailer in y4m format with 480p resolution. The output format was a raw H.264 video stream for all *configurations*.

This case study was executed using different numbers of CPU cores. The *features* 'core1', 'core2', and 'core3' denote *configurations* that were executed using only 1, 2, or 3 CPU cores, respectively. All 4 cores of the CPU were used for *configurations* with the *feature* 'core4'. We used the command `taskset` to 'pin' the process to certain CPU cores.

[2] http://gatling.io/ – last visited on 2017-09-13

5.4 PDU Characteristics

The PDUs of the *Zmiy*-cluster are organized in units of three phases (sockets). The PDU with 18 sockets consists of two parts with three such units each, the small PDU only has one such unit. This layout is depicted in Figure 5.1. The small (red) numbers above each socket denote the number of the cluster node that is connected to the socket. Phase 1 of the unit A1 is defective and two phases are not currently used. In the following, the notation 'A1' will be used to refer to the first unit of part A of the large PDU, 'A1-P1' refers to the first phase of that unit. The colours (borders) of the sockets visualize the results and are described later in this section.

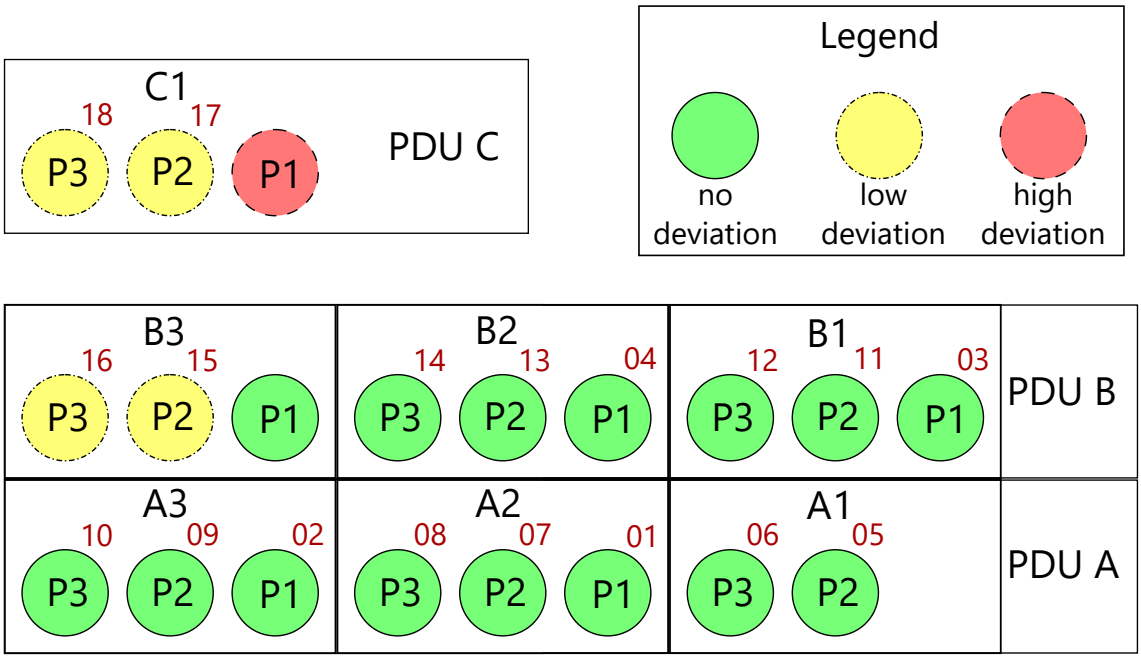

Figure 5.1: Layout of the PDUs from our experimental setup. One PDU consists of two parts A and B with three units each, the other PDU only consists of one part (C) with one unit. Boxes indicate units and circles are sockets (phases). The red numbers above the sockets are the numbers of the connected cluster nodes. The colour (border) of the sockets indicates the deviation of the socket.

Based on previous experiences with the PDUs, we made the assumption that offsets exist between the measured values from the different sockets, or *phases*, of the PDUs. This may be due to the PDU measuring its own power consumption on one of the phases in addition to the power consumption of the connected load.

To be able to use the PDUs for the case studies, these offsets must at least be predictable and the PDUs must meet certain conditions:

Offsets: Offsets between PDUs and PDU phases in the measured values should be equal. Alternatively, they need to be predictable.

Consistency: Constant energy consumption must result in (largely) constant measurements.

Comparability: Equal measurements with different meters should yield the same results.

Repeatability: Equal measurements with the same meter repeated over time must yield the same results.

Isolation: Simultaneous measurements on different phases of the same PDU must not influence one another.

Since, in general, energy measurements can be quite inaccurate under different circumstances ([LKM16]) and to test whether the offsets exist and if these conditions are met, we conducted sanity checks for the PDUs before measuring the case studies.

PDU Sanity Checks Setup

First, the individual phases of the PDUs were measured. The measurements for each phase were conducted with four different load scenarios. For the first scenario, one of the *i5 nodes* was connected to the phase that was measured and no load was connected to all other phases of the PDU. The *i5 node* was idle with a CPU load close to 0%. For the second and third scenario the same *i5 node* was used again, but this time the tool *stress-ng* was used to generate 50% (partial load) and 95% (full load) CPU load, respectively. For the fourth scenario a regular 40W light bulb was used instead of the *i5 node*. A light bulb has an almost constant power consumption and 40W is similar to the power consumption of the used computers with ranges roughly from 20W to 60W.

The three scenarios with an *i5 node* were measured for 15 seconds and 2 minutes, with 10 and 5 repetitions, respectively. For both the short and long measurements, the relative standard deviation between the repetitions was calculated. Measurements with a relative standard deviation greater than 10% were repeated. The fourth scenario was measured for 2 minutes without repetitions, because the power consumption of the light bulb was observed to be fairly constant and no significant deviations between repetitions were to be expected.

To determine a ground truth for the power consumption of all four scenarios, the same scenarios were also measured with a *Watts up?* meter.

The three scenarios with an *i5 node* were measured for the *Watts up?* meter with the same durations and repetitions as for the PDU phases. The light bulb scenario was measured for 10 minutes without repetitions.

Additionally, a test was conducted to examine the influence from load on multiple phases of the same PDU on the measurements. This test was measured exemplarily for only one phase of each PDU.

For the PDU C, the light bulb was connected to phase C1-P1 and two *i5 nodes* running on full load were connected to the other two phases. The power consumption was measured for the phase of the light bulb. For the PDU with 18 sockets, two different scenarios of this test were conducted. Since the PDU is organized in units of three phases each, the same test as for the small PDU was conducted using only the unit B3. Additionally, the test was repeated with full load cluster nodes connected to all 16 other phases.

PDU Sanity Check Results

To compare the results and determine the offsets, the differences of the values measured by the PDU phases to the values measured by the *Watts up?* meter for the same scenario were calculated. Theses differences are visualized in Figure 5.2. On the x-axis are the four different scenarios. The y-axis shows the absolute offsets from the *Watts up?* meter in watts. Each dot represents one PDU phase. Most PDU phases and the *Watts up?* meter measure almost the same values. Those values are assumed to be the true values. Only few phases measure values that deviate by more than 1W in either direction. Three phases were identified that always measure values higher than the true values and two phases always measure values lower than the true values. The phases with deviations are marked in Figure 5.1.

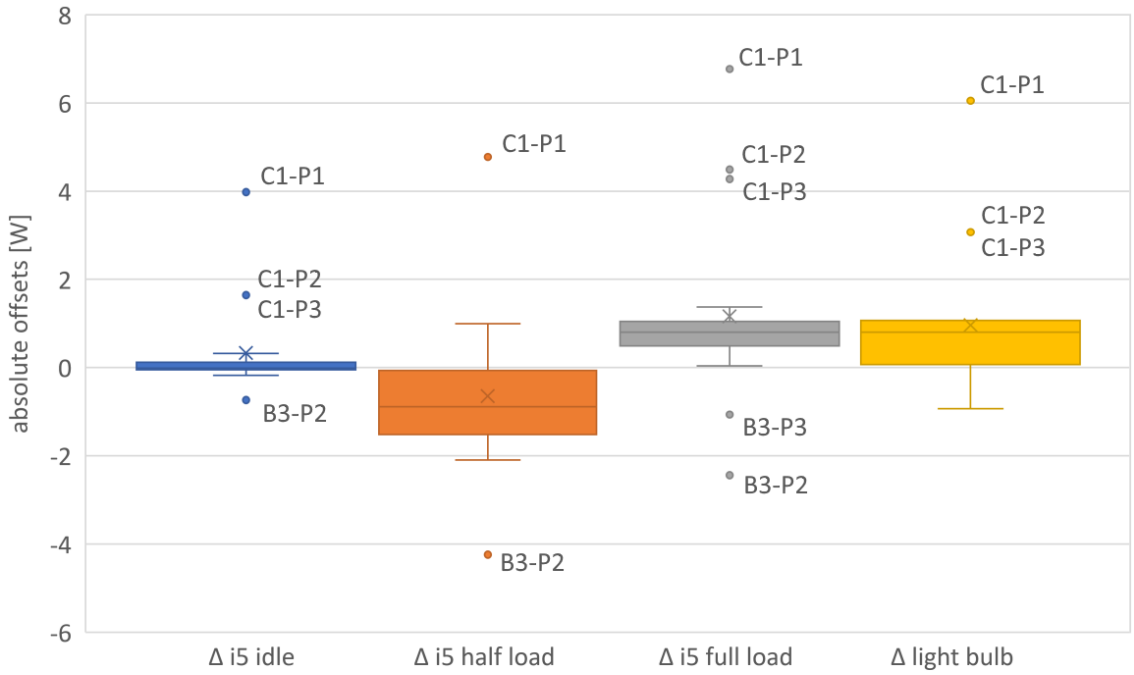

Figure 5.2: Absolute offsets from *Watts up?* in watts. For all four scenarios, each dot represents the offset of one PDU phase.

The predicted offsets exist, but only for five phases.

We then attempted to approximate the offsets as constants and linearly. For that, we calculated the relative standard deviation for each type of measurement between the values of all phases and tried to reduce that deviation as much as possible by subtracting different offsets for those five phases. Both the constant and linear approximations yielded comparable improvements in the deviation. Without subtracting any offsets, the relative standard deviations for the different scenarios ranged from 3.0% to 4.7%. With constant integral offsets, the deviations were between 1.0% and 2.3%. With linear offsets, the deviations were between 0.9% and 2.3%. Since subtracting the linear offsets results in non-integral values while the measured values from the PDUs are only integral, we decided to use the constant offsets. For scenarios with considerably higher power consumption we expect greater differences between using constant and linear offsets but for our measurements both approximations appear equally acceptable.

Using these constant approximations of the offsets, we can use the PDUs to measure the case studies. To make the measurements from all phases comparable, the determined constant offsets are simply subtracted from the values measured during the case studies. Table 5.1 lists the approximated offsets.

Table 5.1: Approximated offsets of the PDU phases

Phase	Offset
B3-P2	-2 W
B3-P3	-1 W
C1-P1	5 W
C1-P2	2 W
C1-P3	2 W

The results of the measurements with light bulb and load on multiple phases were compared to the light bulb test without other loads that was measured on the same phase. The observed differences were less than 1W for all scenarios. Since the resolution of the PDUs is 1W, those differences are likely caused by measurement inaccuracy. Consequently, the PDUs can be used to measure multiple configurations at the same time on different phases and the measurements are not expected to influence one another.

Furthermore, the repetitions of the load tests with *i5 nodes* showed a relative standard deviation of less than 10% and the tests with the light bulb showed virtually constant power consumption measurements. This indicates, that repeating measurements over time always yields approximately the same results.

In the following list we summarize the results with respect to the conditions described above:

Offsets: While the offsets for some phases differ, theses differences are predictable and can be subtracted from the measured values.

Consistency: The constant energy consumption of the light bulb resulted in constant measurements.

Comparability: Except for the phases with different offsets, the PDUs and the *Watts up?* meter measured approximately the same values.

Repeatability: Equal measurements with the same meter repeated over time resulted in the same values with a relative standard deviation of less than 10% between the repetitions.

Isolation: The test with light bulb and load on other phases showed that simultaneous measurements on different phases of the same PDU have no influence on one another.

6. Evaluation

In this chapter, we present the evaluation of the case studies and answer the research questions. We cover one research question in each section in this chapter, beginning each section with the research question. Then, we explain, what metrics we use to answer it, and provide the corresponding results from the case studies. After the presentation of the results, we discuss them and conclude each section by summarizing the answer to the research question.

6.1 General Correlation

In our first research question, we examine the general correlation between *performance* and *energy consumption*:

RQ1: Is there a correlation between *performance* and *energy consumption*?

Operationalization

To answer the question, whether a correlation between *performance* and *energy consumption* exists, we calculate the *Pearson correlation coefficient* for the results from each case study.

Additionally, we plot the results of the case studies with the *performance* on the x-axis and the *energy consumption* on the y-axis, to graphically interpret the results.

Results

In Table 6.1, we show the *Pearson correlation coefficients* for the case studies.

In Figure 6.1, Figure 6.2, and Figure 6.3, we show the results of the *HSQLDB*, *Apache*, and *x264* case studies, respectively. Each dot represents one *configuration*.

Table 6.1: *Pearson correlation coefficients* between *performance* and *energy consumption* for the case studies

Case study	ρ
HSQLDB	0.9070
Apache	0.9082
x264	0.9653

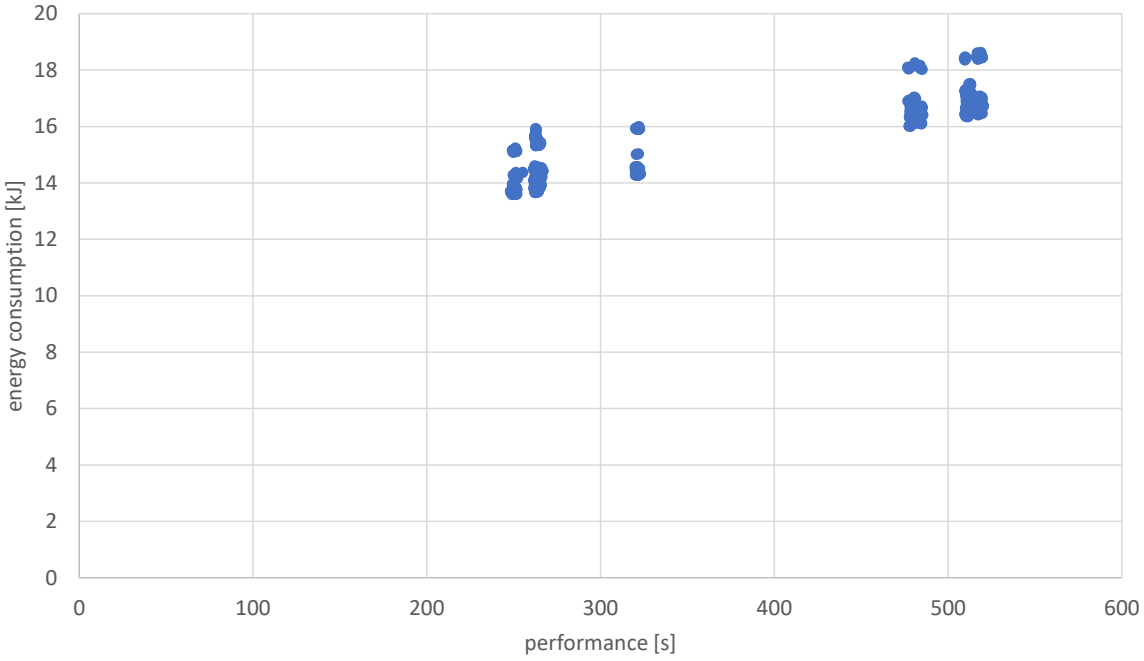

Figure 6.1: Results for *performance* and *energy consumption* of the *HSQLDB* case study. Each dot represents one configuration.

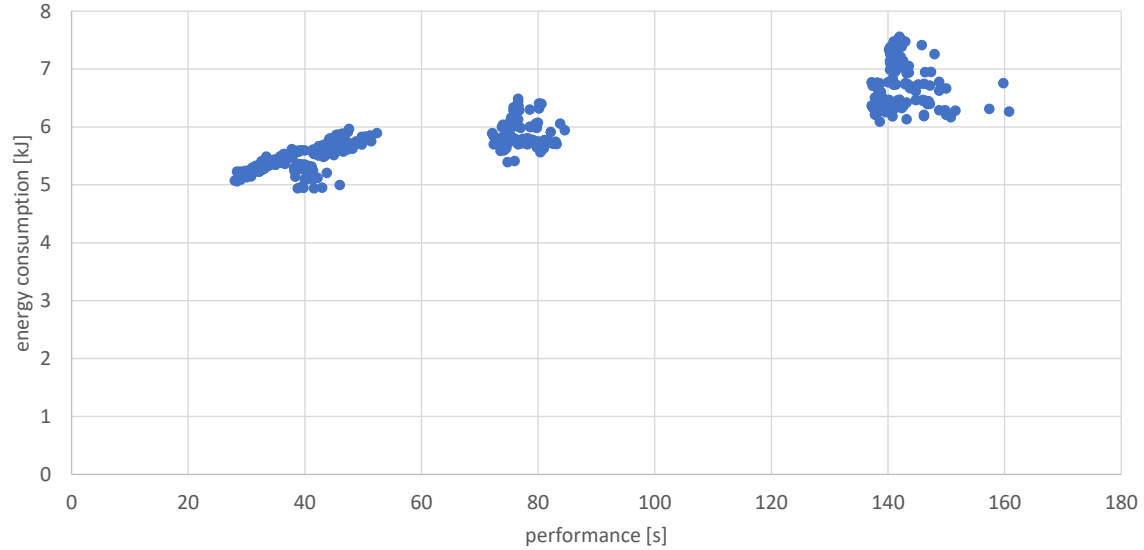

Figure 6.2: Results for *performance* and *energy consumption* of the *Apache* case study. Each dot represents one configuration.

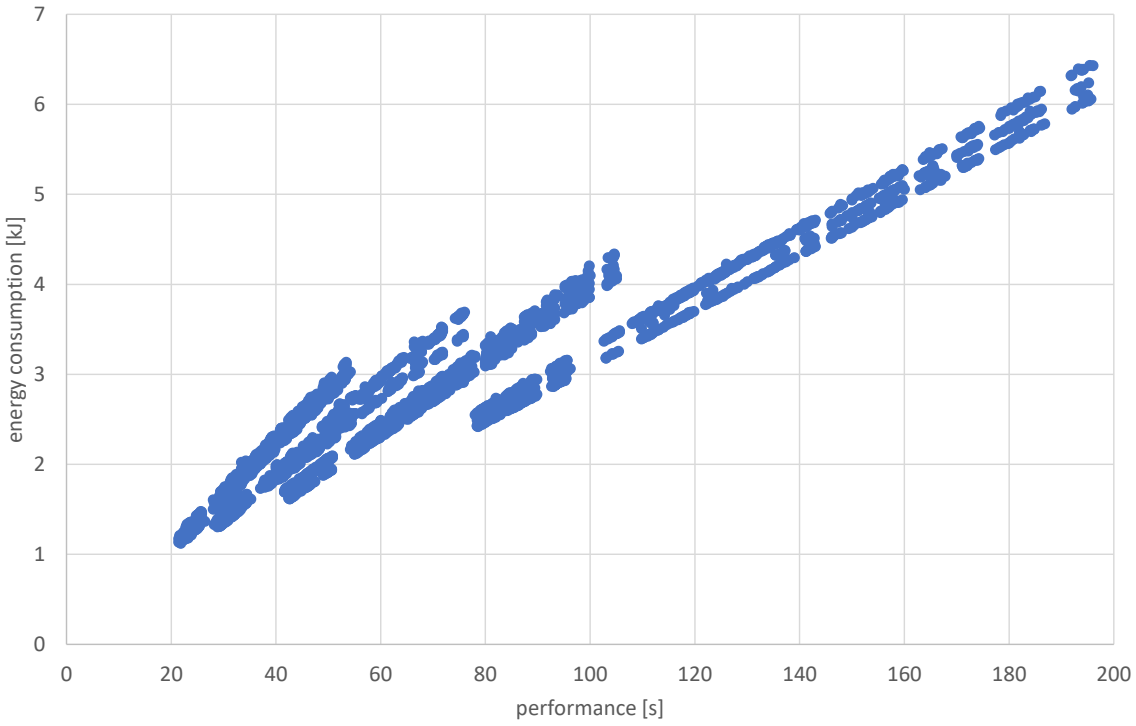

Figure 6.3: Results for *performance* and *energy consumption* of the *x264* case study. Each dot represents one configuration.

Discussion

The *Pearson correlation coefficients* (see Table 6.1) are greater than 0.9 for all three case studies, which indicates a strong linear correlation between *performance* and *energy consumption*.

However, for both the *HSQLDB* and *Apache* case studies, there are configurations with approximately equal *energy consumption* and substantially differing *performance*. For example, the *HSQLDB* case study has *configurations* with an *energy consumption* of approximately 16 kJ and a *performance* of about 260 s and 480 s. Similarly, the *Apache* case study has *configurations* with a *performance* of about 80 s and 150 s which have approximately the same *energy consumption* of 6.5 kJ. In both cases, there is a *configuration* with approximately the same *energy consumption* as another *configuration* and almost double its *performance*.

The same is the case for the *x264* case study. For example, there are *configurations* with an *energy consumption* of approximately 3 kJ and a *performance* of about 50 s and 90 s. However, these two *configurations* were using a different number of CPU cores. When comparing only *configurations* with the same number of CPU cores, there are no two *configurations* with approximately equal *energy consumption* with such a big difference in their *performance*. Here, the difference between the *performance* values for *configurations* with approximately the same *energy consumption* is at most about 15 % of the *performance* of those *configurations*.

Graphically, while, for the *HSQLDB* and *Apache* case studies, a linear correlation between *performance* and *energy consumption* is visible, it does not appear to be very strong. For the *x264* case study, however, the linear correlation is visible in

the plot. Actually, the plot for the *x264* case study shows four distinct lines which correspond to the four *features* that represent different numbers of CPU cores.

Since the correlation between *performance* and *energy consumption* appears to be different for *infrastructure software* and *application software*, we further investigate the correlation independently for *infrastructure software* in Section 6.2 and for *application software* in Section 6.3. For the general correlation, we come to the following conclusion:

> There is a strong linear correlation between *performance* and *energy consumption*. The correlation is stronger for *application software* than for *infrastructure software*.

6.2 Correlation in Infrastructure Software

In the first research question, we investigated the general correlation between *performance* and *energy consumption*. Since the correlation appears to be different for *infrastructure software* and *application software*, we examine the correlation for *infrastructure software* in more detail, which leads to the following research question:

RQ1.1: Is there a correlation between *performance* and *energy consumption* in *infrastructure software*?

Operationalization

To investigate the correlation between *performance* and *energy consumption* specifically for *infrastructure software*, we analyse the plots of *performance* and *energy consumption* from the *HSQLDB* and *Apache* case studies in more detail and compare the *Pearson correlation coefficients* for different sets of *configurations*.

Results

In Table 6.2, we show the *Pearson correlation coefficients* for various sets of *configurations* for the *HSQLDB* and *Apache* case studies.

Table 6.2: Correlation coefficients between *performance* and *energy consumption* for various sets of *configurations* of the *infrastructure software* case studies.

Case study	*Configurations*	ρ
HSQLDB	*(all)*	0.9070
HSQLDB	$\neg$*crypt_blowfish*	0.2508
Apache	*(all)*	0.9082
Apache	$\neg$*servercache*	0.7736

In Figure 6.4 and Figure 6.5 we show the results from the *HSQLDB* and *Apache* case studies in more detail. The plots are enlarged versions of the plots in Figure 6.1 and Figure 6.2, only showing the top right corner while still showing all *configurations*. *Configurations* with *features* that have a dominant influence on both *performance* and *energy consumption* leading to clustering, are highlighted.

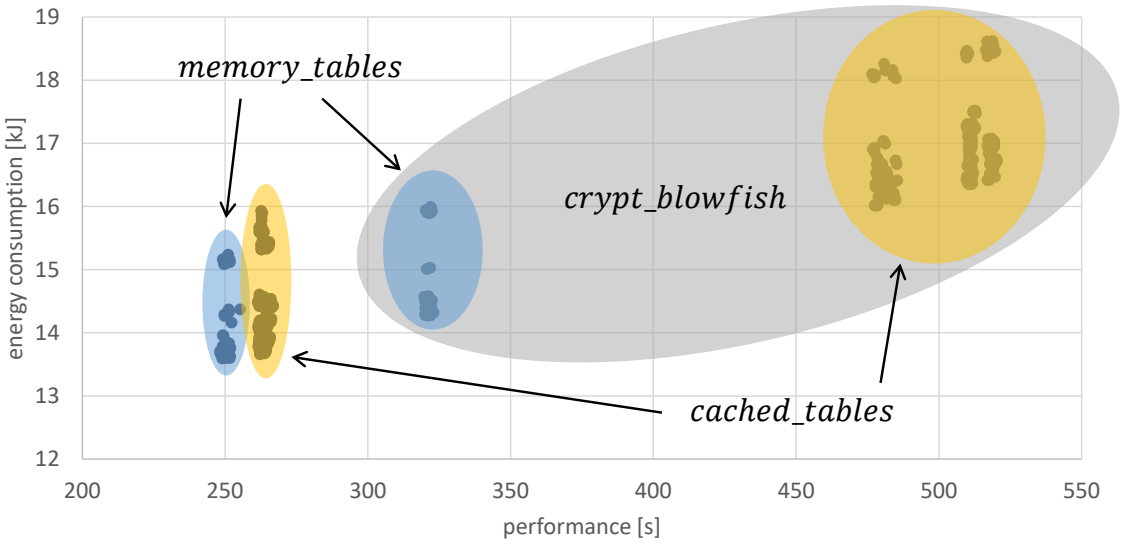

Figure 6.4: Dominant influences from *features* in the *HSQLDB* case study. Each dot represents one configuration. The plot only shows the upper right corner of Figure 6.1. Thus, axes do not start at zero.

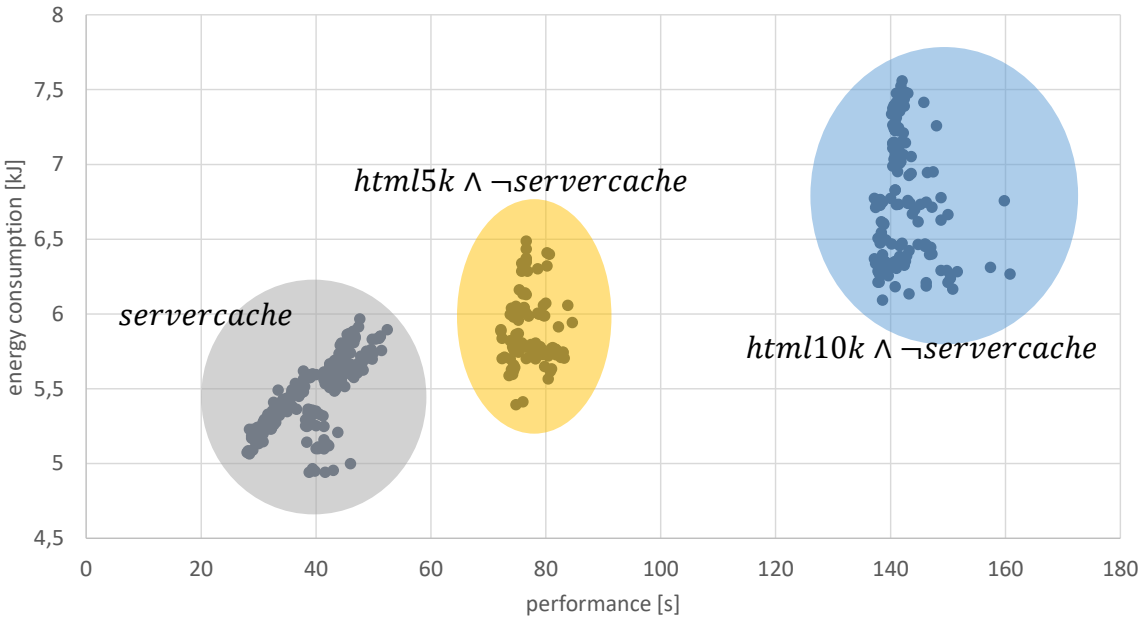

Figure 6.5: Dominant influences from *features* in the *Apache* case study. Each dot represents one configuration. The plot only shows the upper right corner of Figure 6.2. Thus, the y-axis does not start at zero.

Discussion

Both the *HSQLDB* and *Apache* case studies show a strong linear correlation between *performance* and *energy consumption* with $\rho > 0.9$. However, the correlation is only strong when considering many *configurations* with largely varying *performance* and *energy consumption*. Both case studies have one *feature* with a huge influence on both *performance* and *energy consumption* which dominates the linear correlation.

For *HSQLDB*, this *feature* is 'crypt_blowfish' as shown in Figure 6.4. Blowfish encryption is a compute-intensive task and consequently leads to a high energy consumption due to high CPU usage, and takes long, increasing the *performance*.

For *Apache*, it is 'servercache' as shown in Figure 6.5. Using caching on the server results in requests being answered much faster, drastically decreasing the *performance*, because it almost completely avoids slow disk access. Additionally, some computations will not be required when using cached responses. This, in combination with the reduced disk usage, also decreases the *energy consumption*.

In the second research question in Section 6.4, we investigate the influences from individual features in more detail.

For both case studies, leaving out the *feature* with the highest influence on *performance* and *energy consumption* drastically reduces the correlation. For the *HSQLDB* case study, *configurations* without the *feature* 'blowfish' only have a weak linear correlation of $\rho \approx 0.25$. Similarly, for the *Apache* case study, *configurations* without 'servercache' only have a linear correlation of $\rho \approx 0.77$, which is neither weak nor strong.

This indicates, that only one or few *features* with a huge influence on *performance* and *energy consumption* exhibit a correlated behaviour regarding *performance* and *energy consumption*. Most features, however, do not show this correlation.

> The linear correlation between *performance* and *energy consumption* for *infrastructure software* appears to be strong. This strong correlation is caused by certain individual *features* with a dominant influence on both *performance* and *energy consumption*. Excluding such *features* results in a weak linear correlation.

6.3 Correlation in Application Software

In the previous sections, we examined the correlation between *performance* and *energy consumption* in general and specifically for *infrastructure software*. In this section, we examine the correlation in more detail for *application software*:

RQ1.2: Is there a correlation between *performance* and *energy consumption* in *application software*?

Operationalization

To investigate the correlation between *performance* and *energy consumption* specifically for *application software*, we investigate the plot for *performance* and *energy consumption* of the *x264* case study in more detail and compare the *Pearson correlation coefficients* for various sets of *configurations*.

Results

In Table 6.3, we show the *Pearson correlation coefficients* for various sets of *configurations* of the *x264* case study.

Table 6.3: Correlation coefficients between *performance* and *energy consumption* for various sets of *configurations* of the *application software* case study *x264*.

Configurations	ρ
(all)	0.9653
core1	0.9955
core2	0.9956
core3	0.9947
core4	0.9680
$\neg$*no_mixed_refs*	0.9641

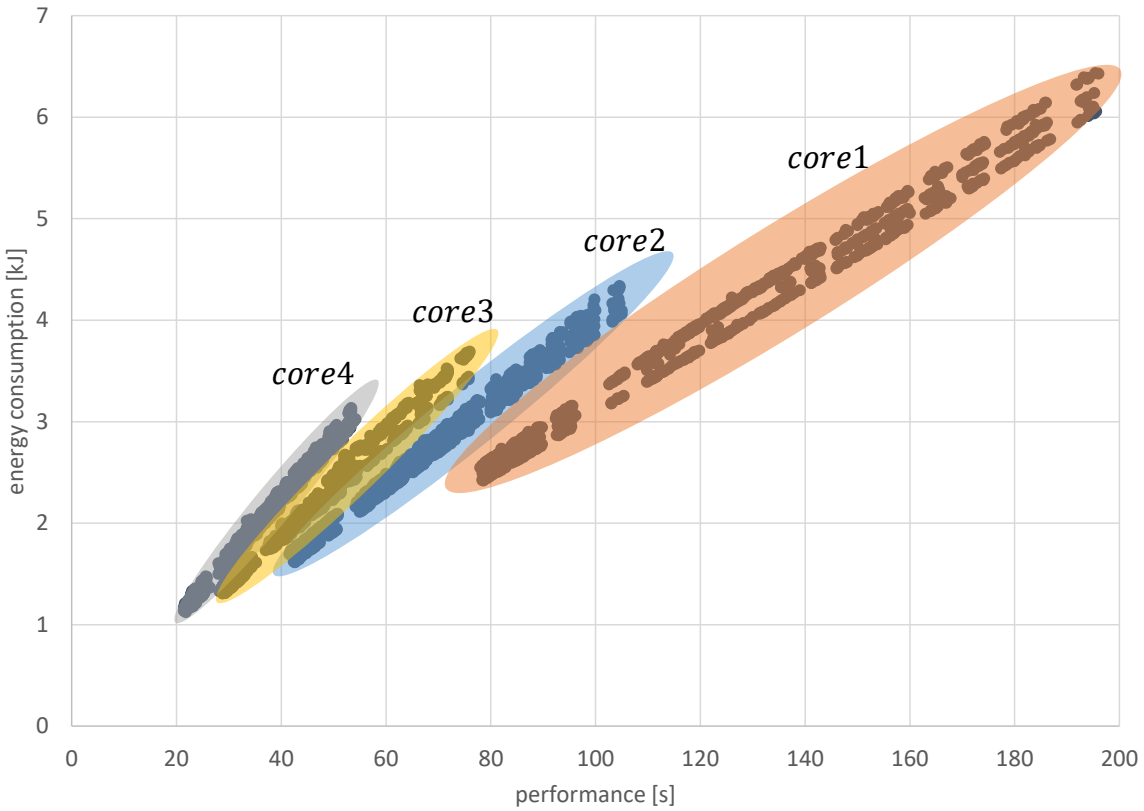

Figure 6.6: Dominant influences from *features* in the *x264* case study. Each dot represents one configuration.

In Figure 6.6, we show the results from the *x264* case study in more detail. *Configurations* with *features* that have a dominant influence on both *performance* and *energy consumption* leading to clustering, are highlighted. Those *features* are those describing differences in the experimental setup using a certain number of CPU cores.

Discussion

By investigating the plot in Figure 6.6, we can see that only the four different variations of the experimental setup with different numbers of CPU cores form clusters. No other *feature* of the case study seems to have a dominant influence on either *performance* or *energy consumption* and, thus, no other *feature* appears to dominate the linear correlation. In fact, not even the 'core' *features* dominate the linear correlation. Rather, there appears to be a strong linear correlation within each cluster.

In Table 6.3, we show the *Pearson correlation coefficients* for the *configurations* using different numbers of CPU cores. The *x264* case study shows a very strong linear correlation between *performance* and *energy consumption* with $\rho > 0.96$. The *configurations* which were restricted to use only certain CPU cores even show an almost perfect linear correlation of $\rho > 0.99$.

By calculating the *Pearson correlation coefficients* for each set of *configurations* leaving out one single *feature*, we can confirm the observation we made from the plot, that no *feature* dominates the linear correlation. The lowest *Pearson correlation coefficient* that is possible for the *x264* case study by leaving out one *feature* is $\rho \approx 0.96$ for *configurations* without the *feature* 'no_mixed_refs'. This is almost the same correlation as for all *configurations* including *configurations* with 'no_mixed_refs'.

The linear correlation between *performance* and *energy consumption* for *application software* is very strong. This strong correlation is not caused by single individual *features*.

6.4 Comparison of Influence Models

In the first research question, we investigated the correlation between *performance* and *energy consumption*. In this research question, we further compare *performance* and *energy consumption* by comparing the *performance-influence models* and *energy-influence models*:

RQ2: Do the *performance-influence models* and *energy-influence models* contain the same terms?

Operationalization

To compare the *performance-influence models* and *energy-influence models*, we first generate the models with *SPL Conqueror*. Since the models cannot be more accurate than the underlying measured data, we calculate an average value (squared average) of the relative standard deviations of our measurements. We use this average as the minimum error rate for the *performance-influence models* and *energy-influence models*.

Then, we compare the models by comparing, which *features* and interactions of *features* appear in the models. To additionally compare the factors in the models,

we normalize the models by dividing each factor by the factor of 'base' (β_0). To aid this comparison of factors, we plot the normalized values.

Since the first research question (see Section 6.3) showed us, that, for the *x264* case study, four similar clusters exist in the results, one for each 'core' *feature*, we examine the *performance-influence models* and *energy-influence models* independently for each 'core' *feature* and compare the models.

Results

In Table 6.5, Table 6.6, and Table 6.7, we show the terms of the *performance-influence models* and *energy-influence models*, ϕ and ϵ, respectively, for each of the case studies. Additionally, the tables contain normalized values for the terms for easier comparison. The normalized values $\hat{\phi}$ and $\hat{\epsilon}$ for ϕ and ϵ, respectively, are obtained by dividing the value by the corresponding value for 'base'.

The normalized values are plotted in Figure 6.7, Figure 6.9, and Figure 6.10. In the plots, the inner area of the circle represents values less than 0 with the centre of the circle being -1. The outer area of the circle represents positive values with the outer line being 1. The two bold lines show the influences in *performance* and *energy consumption* for the *features* and interactions of *features* indicated by the labels around the circle. An empty dot on the bold lines indicates that the corresponding *feature* or interaction is not a part of the model.

For the *HSQLDB* case study, the normalized values are additionally plotted in Figure 6.8. Since this type of plot requires a lot of space, the plots for *Apache* and *x264* are not included here, but are available in Section A.2 of the appendix. The left and right text columns contain the *features* and interactions of *features* with an influence on the *performance* and *energy consumption*, respectively. The coloured area in the plot represents values from -1 on the left to 1 on the right. An empty dot on the bold lines indicates that the corresponding *feature* or interaction is not a part of the model.

In Table 6.4, we show the averages of the relative standard deviations and the errors of the models generated by *SPL Conqueror*. The average deviations are a minimum value for the model error, because the model cannot be more accurate than the data used to generate the model.

Table 6.4: Error rates of the *performance-influence models* and *energy-influence models* and average deviations of the underlying measurement results

Case study	Model	Average deviation	Model error
HSQLDB	*performance*	0.16 %	0.62 %
HSQLDB	*energy consumption*	0.27 %	2.5 %
Apache	*performance*	3.5 %	14 %
Apache	*energy consumption*	0.66 %	2.9 %
x264	*performance* core4	0.53 %	3.8 %
x264	*energy consumption* core4	1.1 %	4.4 %

Table 6.5: Terms of the *performance-influence models* and *energy-influence models* for the *HSQLDB* case study

Features	ϕ	ϵ	$\hat{\phi}$	$\hat{\epsilon}$
base	250	14	1.0	1.0
cached_tables	13	0.20	0.052	0.014
crypt_blowfish	248	0.89	0.99	0.063
memory_tables × crypt_blowfish	-179		-0.72	
memory_tables × crypt_blowfish × txc_locks	30		0.12	
crypt_blowfish × txc_locks	-32		-0.13	
cached_tables × crypt_blowfish		1.9		0.13
no_write_delay		-0.34		-0.024
large_cache		0.47		0.034
large_cache × small_log		-0.47		-0.033

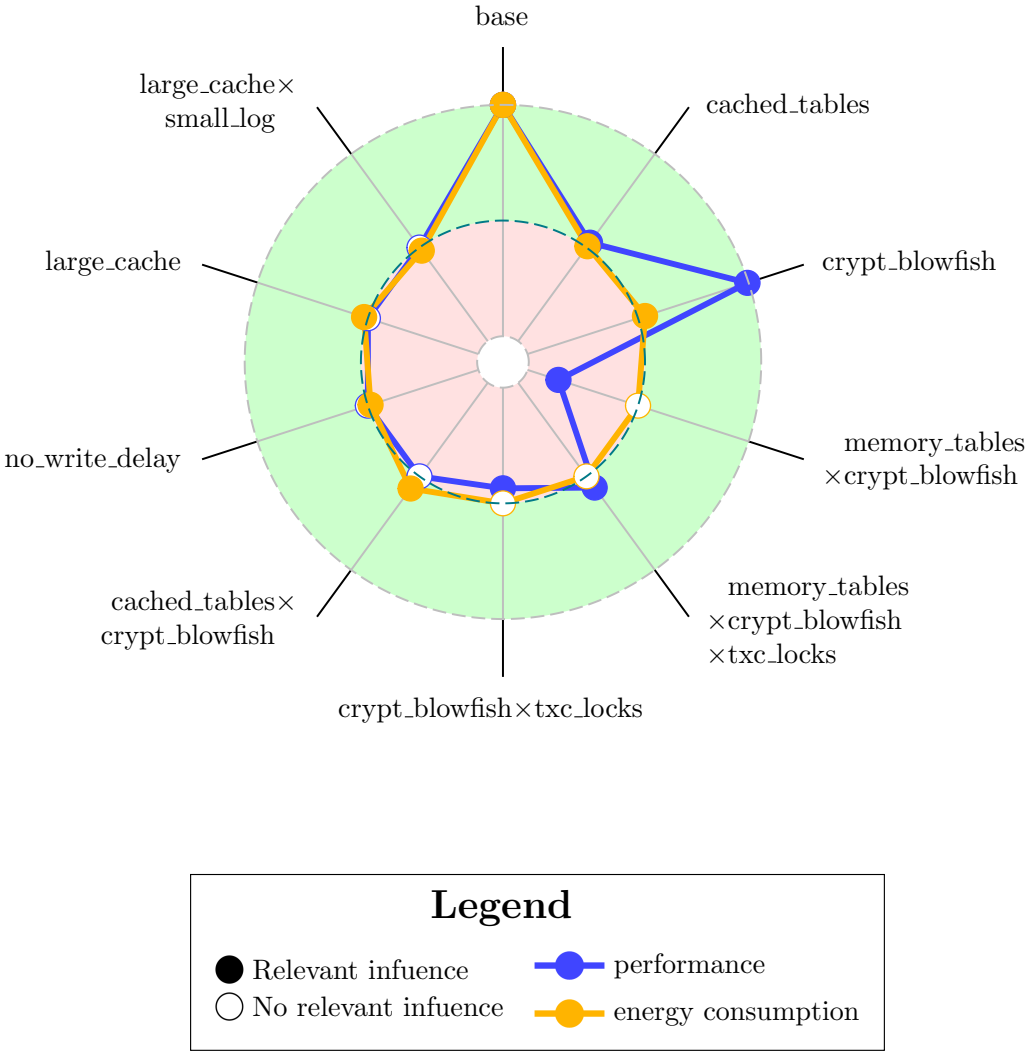

Figure 6.7: Normalized terms of the *performance-influence models* and *energy-influence models* for the *HSQLDB* case study

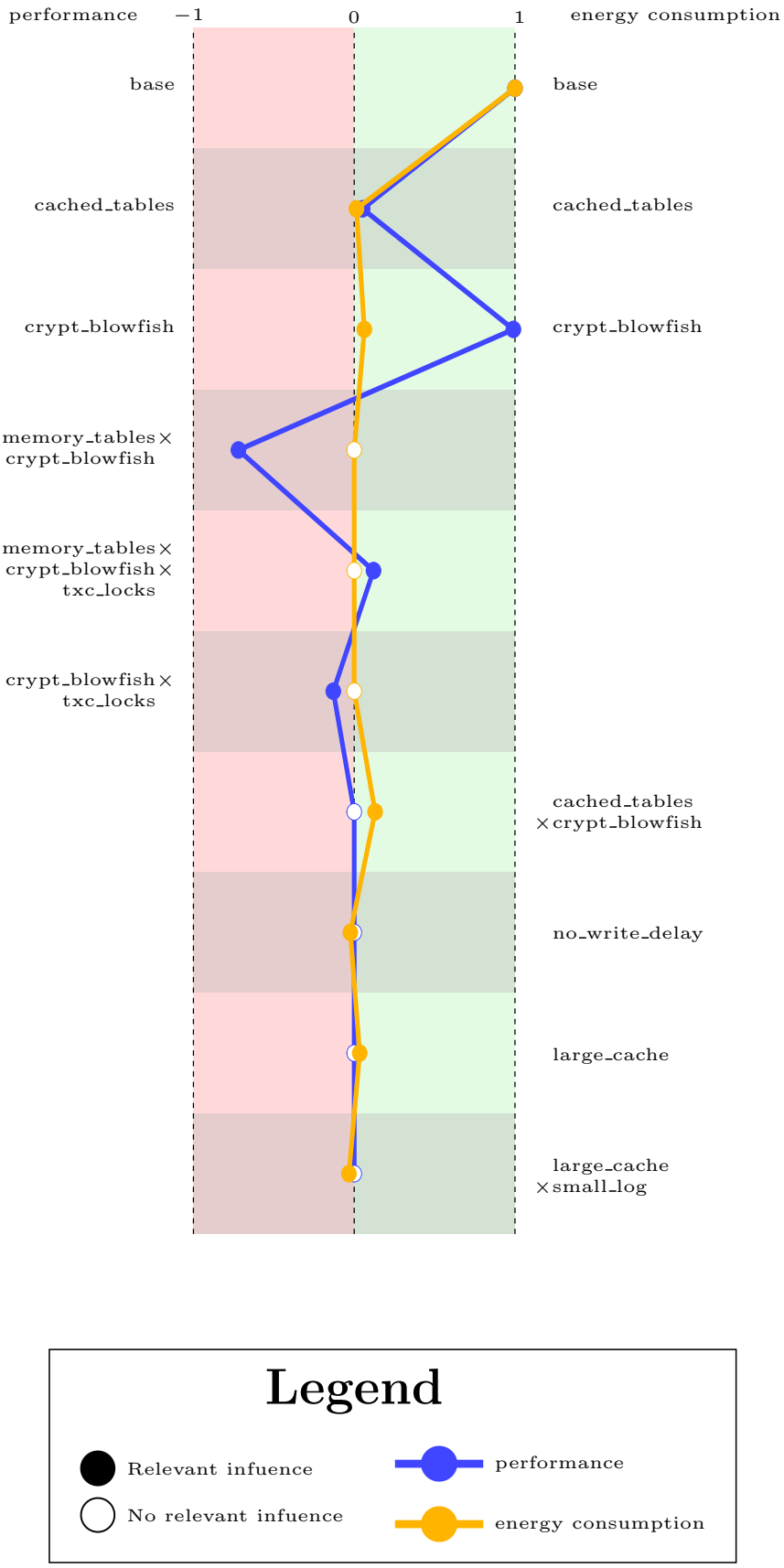

Figure 6.8: Normalized terms of the *performance-influence models* and *energy-influence models* for the *HSQLDB* case study

Table 6.6: Terms of the *performance-influence models* and *energy-influence models* for the *Apache* case study

Features	ϕ	ϵ	$\hat{\phi}$	$\hat{\epsilon}$
base	125	7.0	1.0	1.0
compression	8.4	0.17	0.068	0.024
servercache	-82	-1.4	-0.65	-0.20
keepalive	-18	-1.1	-0.15	-0.15
keepalive × servercache	9.3	0.55	0.074	0.079
keepalive × worker	-3.5		-0.028	
event	-1.1		-0.0089	
servercache × compression	-6.1		-0.048	
prefork	3.5		0.028	
keepalive × html10k		0.33		0.047

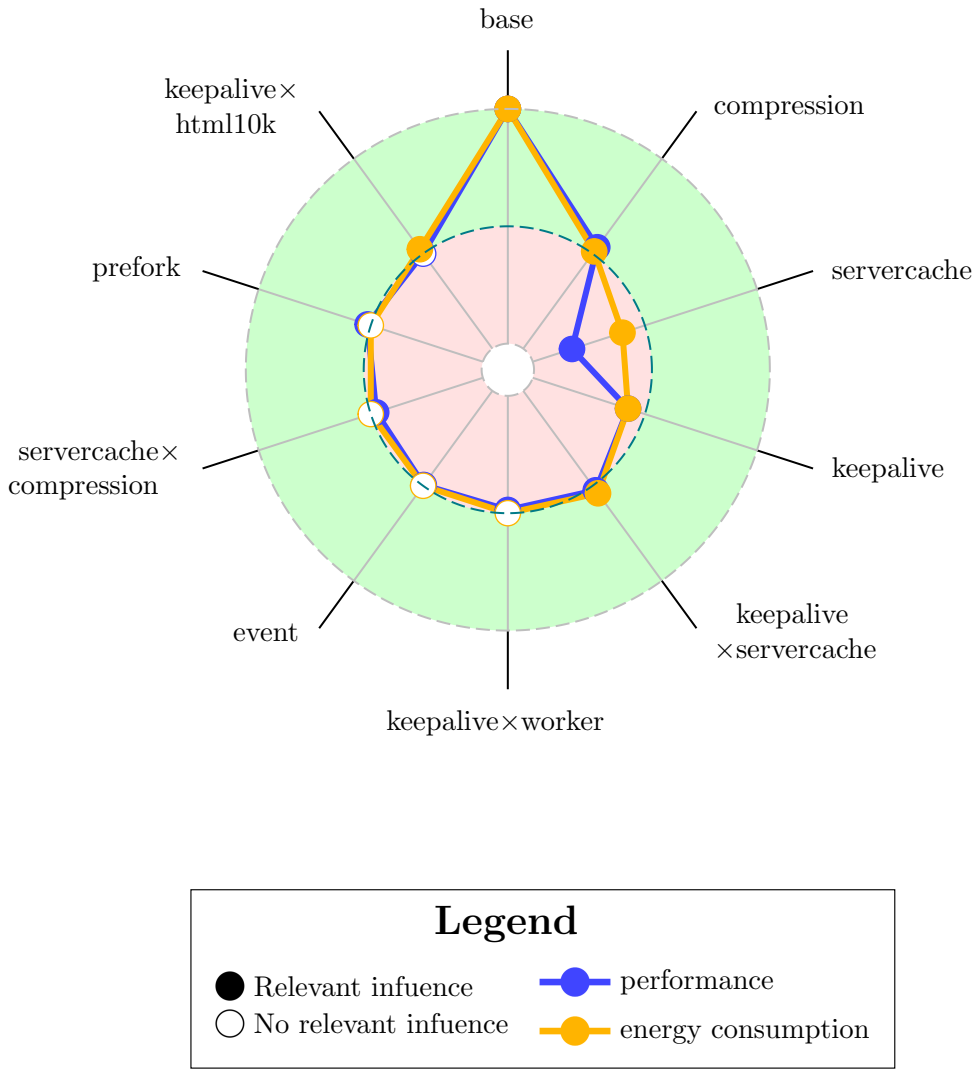

Figure 6.9: Normalized terms of the *performance-influence models* and *energy-influence models* for the *Apache* case study

Table 6.7: Terms of the *performance-influence models* and *energy-influence models* for *configurations* with four CPU cores of the *x264* case study

Features	ϕ	ϵ	$\hat{\phi}$	$\hat{\epsilon}$
base	45	2.5	1	1
no_8x8_dct	-1.5	-0.10	-0.032	-0.040
ref_1	-21	-1.2	-0.47	-0.49
ref_5	-11	-0.61	-0.24	-0.24
ref_5 × no_mixed_refs	-4.1	-0.23	-0.091	-0.093
no_mbtree	1.3	0.12	0.028	0.047
ref_5 × no_fast_pskip	2.1	0.12	0.047	0.049
rc_lookahead_20	-0.65		-0.014	
rc_lookahead_60		0.020		0.0081

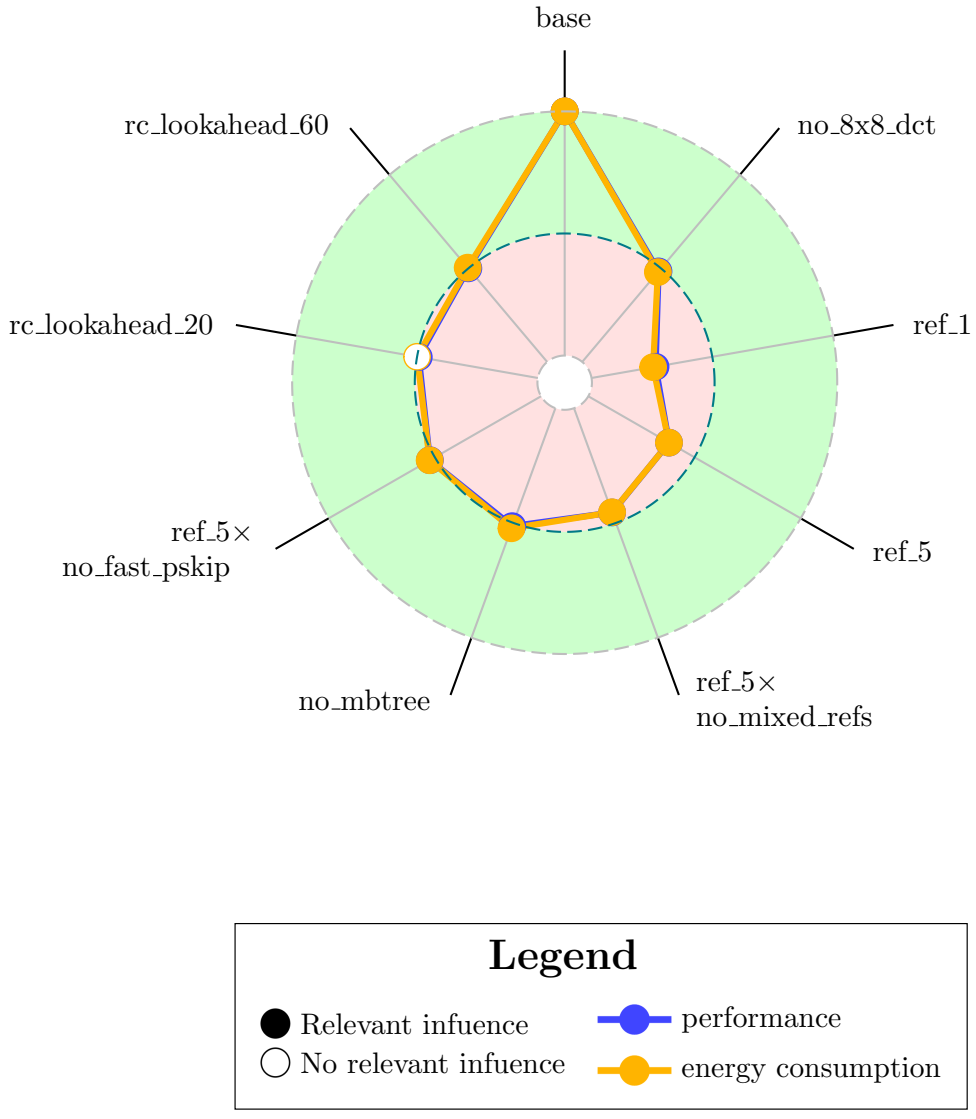

Figure 6.10: Normalized terms of the *performance-influence models* and *energy-influence models* for *configurations* with four CPU cores of the *x264* case study

Discussion

The models show, that, for the *HSQLDB* case study, the *feature* 'crypt_blowfish' has a large impact on both *performance* and *energy consumption*. This is the same *feature* that we already identified in the first research question as the one *feature* having a dominant influence. Another high influence comes from the interaction of the *features* 'memory_tables' and 'cached_tables' with 'crypt_blowfish'. The interaction with 'memory_tables' appears only in the *performance-influence model* and the interaction with 'cached_tables' appears only in the *energy-influence model*. Initially, the influences of these interactions seem to be the largest difference between the *performance-influence model* and *energy-influence model*. However, the influence of the interaction with 'memory_tables' is negative and the influence of the interaction with 'cached_tables' is positive and the two *features* are mutually exclusive. Hence, the influences from those interactions are actually very similar.

The feature 'crypt_blowfish' is very compute intensive resulting in longer executing time with high CPU load, which has a large influence on both *performance* and *energy consumption*. The other encryption *feature* – 'crypt_aes' – has a much smaller influence because of hardware support for AES encryption (*Intel AES New Instructions*). Whether the *feature* 'cached_tables' or 'memory_tables' is used, has a high impact on both *performance* and *energy consumption* because using 'cached_tables' results in much more disk access. This difference in influences mainly occurs in the interaction with 'crypt_blowfish', because *HSQLDB* only encrypts data that is written to the disk, which means that with 'memory_tables' almost no encryption takes place.

There are, however, considerable differences in the *performance-influence model* and *energy-influence model* for *HSQLDB*. Several *features* only have an influence on either *performance* or *energy consumption*. For example, interactions with 'txc_locks' only have an influence on the *performance* and the *features* 'no_write_delay' and 'large_cache' only have an influence on the *energy consumption*.

In a similar way, the *performance-influence model* and *energy-influence model* for *Apache* have similarities and differences. While the *features* 'servercache' and 'keepalive' have a large influence on both *performance* and *energy consumption*, many *features* like 'event', 'prefork', or 'html10k' only influence either the *performance* or *energy consumption*.

The largest influences on *performance* and *energy consumption* are caused by the following *features*: The 'servercache' *feature* drastically reduces the number of disk accesses and the *feature* 'compression' is compute intensive. With the 'keepalive' *feature*, reusing connections reduces the overhead from opening and closing connections.

Furthermore, for both *HSQLDB* and *Apache*, all influences on the *energy consumption* are considerably smaller than on the *performance*. This is likely caused by the idle energy consumption that is measured for all *configurations*.

Both the results from the *HSQLDB* and *Apache* case studies indicate, that, for *infrastructure software*, while some *features* have a high influence on both *performance*

and *energy consumption*, there are considerable differences between the terms in the *performance-influence models* and *energy-influence models*.

For the *x264* case study, we expect that the *features* restricting the number of CPU cores have the largest influence on both *performance* and *energy consumption*. Additionally, we expect that the influences from all other *features* would vary with the number of CPU cores, leading to a *performance-influence model* and *energy-influence model* with many terms for interactions with 'core' *features*. For that reason, we generate the models independently for four sets of *configurations*, one for each 'core' *feature*.

We show only the model for four CPU cores in Table 6.7 and Figure 6.10. The models for one, two, and three CPU cores are very similar to the model for four cores. With normalized factors, all four models are approximately equal.

For the *x264* case study, the *performance-influence model* and *energy-influence model* are very similar. Not only do they contain the same terms, save for the 'rc_lookahead' *features* with an influence of only about 1% which is less than the error rate of the models. The influences are also almost the same for both *performance* and *energy consumption*. In fact, the normalized *performance-influence model* and *energy-influence model* are almost the same.

The graphical representation of the models impressively shows how similar the *performance-influence model* and *energy-influence model* are: In the plot in Figure 6.10, it is barely possible to see the line for the *performance*, because it is 'behind' the line for the *energy consumption*.

For *infrastructure software*, *performance-influence models* and *energy-influence models* have terms for *features* with a dominant influence on *performance* and *energy consumption* in common, but differ otherwise. For *application software*, the *performance-influence models* and *energy-influence models* are very similar.

7. Validity

7.1 Internal Validity

This section lists potential sources of errors and restrictions in our experimental setup, that might lead to inaccurate or invalid results, and explains measures taken to mitigate these threats to validity.

CPU

Modern CPUs have several features that attempt to improve the performance or reduce the energy consumption under certain conditions. These conditions are virtually impossible to predict and could differ between multiple measurements.

Software that is executed on multiple CPU cores may not always use the same cores but the CPU can dynamically assign the software to different cores. This can lead to cores frequently switching from idle to an active state and can have an unpredictable impact on the *energy consumption* and *performance*.

Two such features supported by the CPUs from our experimental setup are *Intel Turbo Boost*[1] and *Enhanced Intel SpeedStep*[2]. *Turbo Boost* can increase the frequency of individual CPU cores under certain conditions when only few cores are used. *SpeedStep* allows the operating system and applications to dynamically adjust the frequency of the CPU to reduce energy consumption.

We have disabled both these features for our measurements.

For software that is intended to run on only some cores of the CPU, we assigned specific cores to the running process to reduce the influence from switching cores.

[1] https://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html – last visited on 2017-08-31

[2] http://download.intel.com/design/network/papers/30117401.pdf – last visited on 2017-08-31

Background Software

Software running in the background during a measurement, such as automatic updates, can influence the *performance* and especially the *energy consumption*.

The minimal installation of the operating system in our experimental setup ensures that only very few background applications are present. Most of the time, the CPU load from background applications is less than 1% and often even measured as 0% (with the command *top*). If they still influence a measurement, the influence happens very likely only during a short time and can thus be detected by repeating measurements and comparing the results (see paragraph 'Repetitions and Deviation').

Measurement Setup

One piece of software that has to run during the measurements, is *PDUGetterV2* (see Section 5.2), which periodically queries the power consumption values from the PDUs .

To make sure that *PDUGetterV2* does not influence the measurements, another node from the cluster is allocated for this tool in addition to the nodes used to run each configuration.

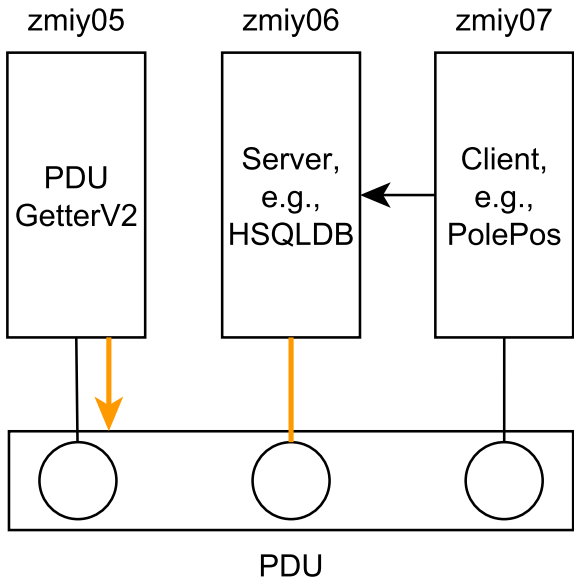

Figure 7.1: Experimental setup with multiple nodes. Three cluster nodes are connected to a PDU. The server and client node communicate over the network. The *PDUGetterV2* runs on a separate node and queries the server node's power consumption from the PDU.

We illustrate an example measurement setup with multiple nodes in Figure 7.1. 'zmiy05', 'zmiy06', and 'zmiy07' are three of the *i5 nodes* from our cluster which are connected to three sockets of a PDU. *PDUGetterV2*, the measured *infrastructure software* and the benchmark client run on three different nodes and, consequently, cannot influence one another's *performance* or *energy consumption*. The arrows

indicate that the client and the server communicate over the network and that *PDUGetterV2* queries the power consumption values from the PDU. In this example, only the power consumption of the server node is relevant (indicated by a bold, coloured line).

PDUs

The PDUs could be the source of invalid measurements just as well as the hardware or software of the nodes. Measurements could yield different results when repeated over time or when measured on different phases ('sockets') of the PDUs. Simultaneous measurements on different phases could influence one another and result in different measured values.

In Section 5.4, we investigated whether the PDUs used in our experimental setup cause any such invalid measurements, motivated by previous experiences with the PDUs, and came to the conclusion that this is not the case.

Network

For the case studies with a server–client setup, the network connecting the nodes running the server and the client might be a limiting factor. In that case, the *performance* measured would mainly be the *performance* of the network, rather than the *performance* of the server node which should be measured.

All user files on the cluster node are located in a network file system. Executing the measured software or the benchmark from the network file system could impact the *performance* of measurements.

In our experimental setup, we cannot rule out that the network does in fact greatly influence the measurements, but we expect that the influence of the network is approximately the same for all measured *configurations*. Since the CPU load of the server during the measurements is still substantially higher than 0%, we expect that the server is not only waiting for the network and we still at least partially measure the *performance* of the server itself.

To avoid the impact from the network file system on *performance*, the files required for the measurement are copied to the local disk of the cluster node prior to the execution of the measurement.

Warm-Up

Applications running on a cluster node directly before a measurement – or the absence of such an application – can influence the measurements. Especially the temperature of the CPU can influence the *performance* as stated by Mytkowicz et al. [MDHS09].

To ensure equal initial conditions for all measurements, a warm-up phase for the CPU precedes all measurements. The warm-up phase uses the tool *stress-ng*[3] to maintain a CPU load of about 95% for 30 seconds.

[3] http://kernel.ubuntu.com/~cking/stress-ng/ – last visited on 2017-08-20

Repetitions and Deviation

All the above measures can not completely rule out unpredictable influences. They also cannot ensure that repeated measurements will yield the same results.

To detect such influences and ensure that repeated measurements actually produce (approximately) equal results, all *configurations* are measured 5 times and the relative standard deviation between those 5 repetitions is calculated. The relative standard deviation is calculated as the absolute standard deviation divided by the mean value. If the relative standard deviation for *performance* or *energy consumption* is greater than 10%, the measurements for the *configuration* are repeated.

7.2 External Validity

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

With *HSQLDB* and *Apache*, we measured two examples for *infrastructure software* from different domains – a database system and a web server. Considering that the results for both case studies were very similar, we assume that other examples for *infrastructure software* have a similar behaviour regarding *performance* and *energy consumption*. However, only two case studies cannot be enough to generalize our findings for all *infrastructure software*.

For *application software*, *x264* was the only case study we conducted. We expect most *application software* to have a strong correlation between *performance* and *energy consumption* because *application software* can usually fully utilize the available resources. On the other hand, the *performance* and *energy consumption* of *infrastructure software* can vary over time because it depends on the presence and needs of clients. Nevertheless, only one case study cannot be enough to conclude that our findings are applicable to all types of *application software*.

To be able to generalize our results we would need case studies from multiple domains for both *infrastructure software* and *application software*. This was not possible within the time frame available for a bachelor thesis.

8. Conclusion

8.1 Summary

In this thesis, we investigated performance and energy interactions of configurable systems. More specifically, we explored a possible correlation between *performance* and *energy consumption* and analysed differences in the *performance-influence model* and *energy-influence model* generated by *SPL Conqueror*.

Before we could measure and evaluate case studies to answer our research questions, we had to conduct sanity checks for our measurement setup. We found that the measurement setup is suitable for the case studies.

Afterwards, we conducted three case studies – *HSQLDB*, *Apache*, and *x264* – and used the results to answer our research questions.

In the first research question, we analysed the correlation between *performance* and *energy consumption*. We found, that *performance* and *energy consumption* are linearly correlated, and the correlation is weak for *infrastructure software*, but strong for *application software*.

In the second research question, we compared the *performance-influence models* and *energy-influence models* that we obtained from the measurement results of the case studies. We found that the models for *infrastructure software* contain different influences for *performance* and *energy consumption*, while the *performance-influence models* and *energy-influence models* for *application software* are very similar.

8.2 Future Work

In this section, we outline possible future work based on this thesis.

In Section 7.2 we explained, that we cannot generalize our results because we only measured a very limited number of three case studies. In future work, the same comparisons we conducted could be applied to more case studies from different domains to generalize the results.

We measured the energy consumption of the whole system. Future work could extend the measurement setup by measuring the energy consumption of individual hardware components like in the study by Tsirogiannis et al. [THS10]. This would allow a more detailed analysis of influences from *features*, which might only impact the energy consumption of some hardware components but not of others.

In our first research question (see Section 6.2), we found that for *infrastructure software* some *features* cause a linear correlation between *performance* and *energy consumption*, while others do not. We argued, that this is caused by a comparably huge influence on both *performance* and *energy consumption*. In future work, the difference between such *features* with and without correlation could be further analysed, by, for example, involving the developers of the software systems.

A. Appendix

A.1 Content of the Accompanying CD

The accompanying CD contains the files necessary to conduct the case studies and the results of the case studies. The *Sintel* trailer used as benchmark for *x264* is not included on the CD, because it is too large for a CD. At the time of this writing (2017-10-09), it is available for download on https://media.xiph.org/.

The following files and directories are included on the CD:

thesis.pdf: This thesis as PDF file.

PDUGetterV2.zip: Source code of *PDUGetterV2*. A compiled JAR file is included in **energymetering.zip**.

energymetering.zip: Shell script used to run the case studies. Includes additional files required to use the script.

hsql: Contains the following files from the *HSQLDB* case study:

- **hsqldb.zip:** The *HSQLDB* database.
- **polepos.zip:** The *PolePosition* benchmark.

apache: Contains the following files from the *Apache* case study:

- **httpd-2.4.25.zip:** Source code of *Apache* with included *APR* and *APR-Util*.
- **gatling.zip:** The *Gatling* load testing tool.

x264: Contains the following files from the *x264* case study:

- **x264:** The *x264* binary.

In addition to the files listed above, each of the directories for the case studies also contains the following files:

featuremodel.xml: Feature model the case study for *SPL Conqueror*.

configurations.list: Text file containing all measured *configurations*, one in each line.

splc.csv: Results of the case study as CSV file for *SPL Conqueror*.

run.sh: Shell script used to start the measurements for the case studies. The directory contains several additional files used by this script.

A.2 Additional Plots

This section contains additional plots that were omitted from Section 6.4 for brevity. Analogous to the plot in Figure 6.8, the plots in Figure A.1 and Figure A.2 visualize the *performance-influence models* and *energy-influence models* for the *Apache* and *x264* case studies, respectively.

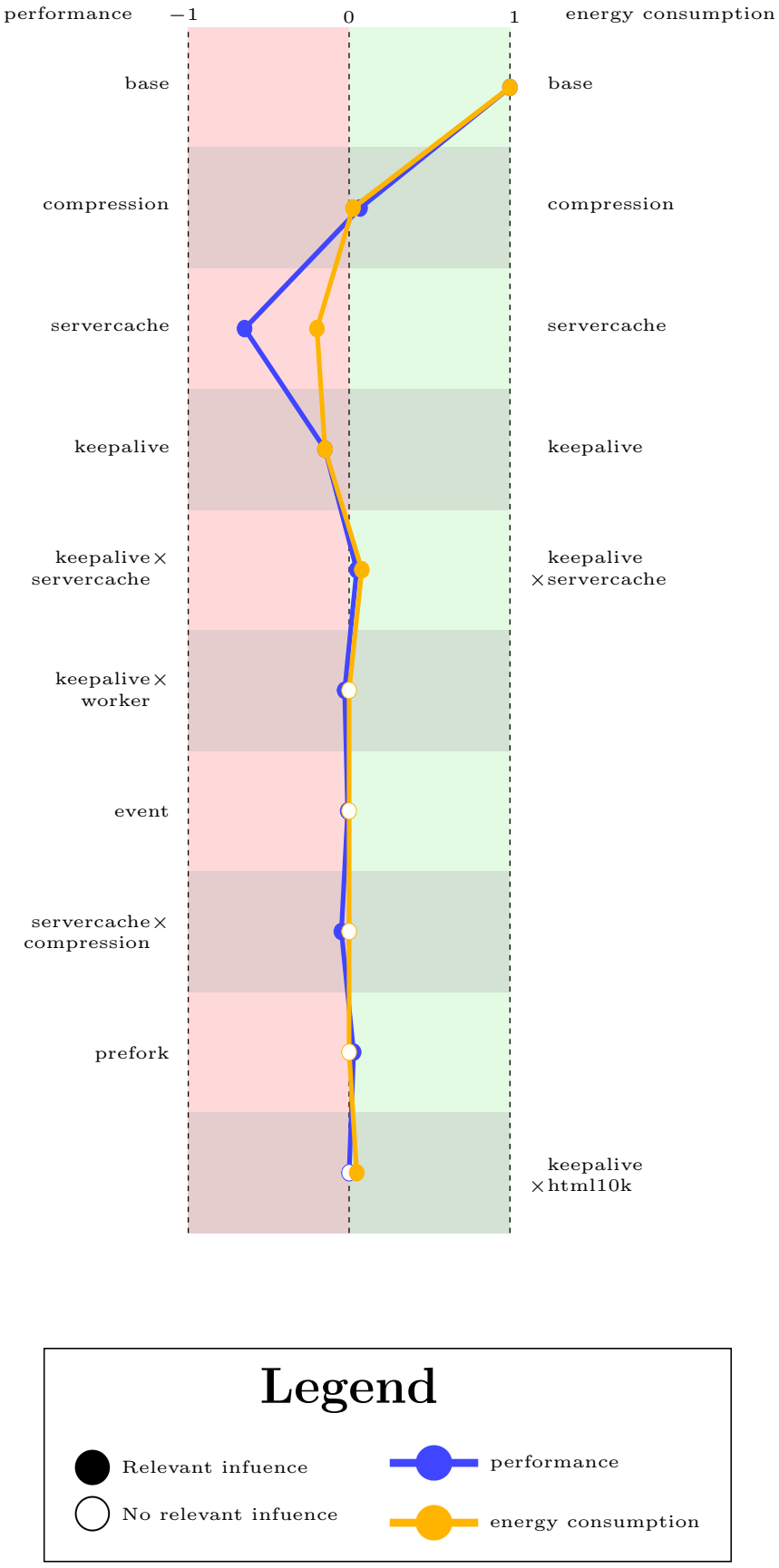

Figure A.1: Normalized terms of the *performance-influence models* and *energy-influence models* for the *Apache* case study

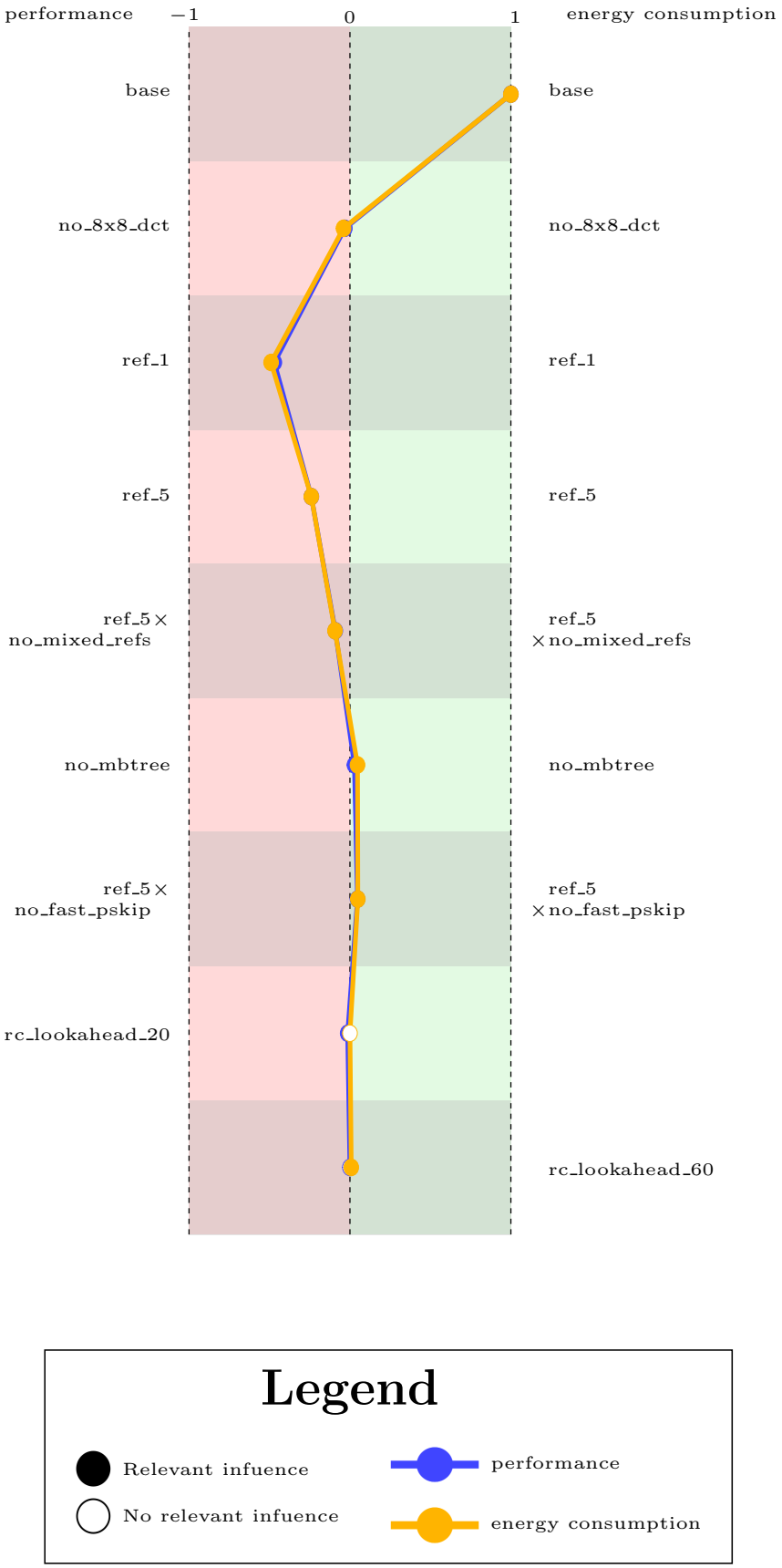

Figure A.2: Normalized terms of the *performance-influence models* and *energy-influence models* for *configurations* with four CPU cores of the *x264* case study

Bibliography

[BCHC09] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. *Pearson Correlation Coefficient*, pages 1–4. Springer Berlin Heidelberg, 2009. (cited on Page 11)

[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*. IEEE, 2011. (cited on Page 3 and 4)

[GBE07] Andy Georges, Dries Buytaert, and Lieven Eeckhout. Statistically rigorous java performance evaluation. *ACM SIGPLAN Notices*, 42(10):57–76, 2007. (cited on Page 20)

[LKM16] Frank Leferink, Cees Keyer, and Anton Melentjev. Static energy meter errors caused by conducted electromagnetic interference. *IEEE Electromagnetic Compatibility Magazine*, 5(4):49–55, 2016. (cited on Page 23)

[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 1 and 45)

[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 11)

[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*, pages 284–294. ACM, 2015. (cited on Page 10 and 11)

[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*, pages 231–242. ACM, 2010. (cited on Page 3, 4, 7, and 48)

[XTW10] Zichen Xu, Yi-Cheng Tu, and Xiaorui Wang. Exploring power-performance tradeoffs in database systems. In *IEEE International Conference on Data Engineering*, pages 485–496. IEEE, 2010. (cited on Page 3 and 4)

Eidesstattliche Erklärung:

Hiermit versichere ich an Eides statt, dass ich diese Bachelorarbeit 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 Bachelorarbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Niklas Werner

Passau, den 11. Oktober 2017