Energy Efficiency of ORM Approaches: an Empirical Evaluation

Giuseppe Procaccianti
Vrije Universiteit Amsterdam
Dept. of Computer Science
Amsterdam, the Netherlands
g.procaccianti@vu.nl

Patricia Lago
Vrije Universiteit Amsterdam
Dept. of Computer Science
Amsterdam, the Netherlands
p.lago@vu.nl

Wouter Diesveld
Diesveld Query Technology
Amsterdam, the Netherlands
wouter@querytechnology.com

ABSTRACT

Context. Object-Relational Mapping (ORM) frameworks are widely used in business software applications to interact with database systems. Even if ORMs introduce several benefits when compared to a plain SQL approach, these techniques have known disadvantages.

Goal. In this paper, we present an empirical study that evaluates the energy efficiency of three different approaches to programmatically access SQL databases in PHP applications. The selected approaches are: plain SQL queries in the source code, and two specialized frameworks, Propel and TinyQueries.

Method. We performed an empirical experiment in a controlled environment. We selected three factors for our experimentation: the different ORM approaches, the type of query (Create, Read, Update, Delete) and the size of database tables. Our response variables were execution time and energy consumption.

Results. As expected, pure SQL yielded the best performance and energy efficiency in all test cases. Propel exhibited a much higher energy consumption and longer execution times. The TinyQueries tool performed slightly worse than SQL, but significantly better than Propel, offering a convenient trade–off between ORM benefits and energy efficiency.

Conclusions. Our experiment shows that ORM approaches have a significant impact on both energy consumption and performance. This helps developers and architects when considering the trade-off between their benefits (e.g. in terms of code maintainability and readability) and drawbacks.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*performance*

Keywords

Energy Efficiency, Object-Relational Mapping, Empirical Experiment

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ESEM '16, September 08-09, 2016, Ciudad Real, Spain
Copyright 2016 ACM X-XXXXX-XX-X/XX/XX ...$15.00.

1. INTRODUCTION

Power consumption is nowadays constraining any type of IT device, from mobile phones to supercomputers. IT energy consumption keeps rising, now reaching almost 5% of worldwide electricity consumption (920 TWh) [5]. Data centers are responsible for about one-third of this amount and reducing their energy impact is more and more a priority not only for environmental reasons, but also (if not mostly) for their economic sustainability: operational electricity costs account for 20% of a datacenter Total Cost of Ownership (TCO) [7]. As a matter of fact, hardware technologies witnessed significant improvements in terms of energy efficiency. Koomey's law states that the number of computations per watt of a computing device doubles every 1.5 years [10]. However, software can be hardly defined efficient in its resource usage. Already in 1995, Niklaus Wirth [29] was advocating against the progressive decrease in software efficiency, as hardware technologies became cheaper. Nowadays, we find ourselves compelled to update our hardware devices *because of software inefficiency*, sometimes without any significant improvement in terms of features or quality.

Traditionally, energy efficiency has been mostly a concern of hardware experts and very specific computing fields, such as embedded systems or high-performance computing, due to the peculiar energy requirements of such contexts. Recently, the Software Engineering community has defined energy efficiency has a relevant, general property of software [11, 12, 8]. In this regard, we envision the role of researchers in software energy efficiency as providing guidelines to address software energy efficiency concerns, based on sound empirical evidence. In particular, this study aims at evaluating the energy impact of Object-Relational Mapping (ORM) frameworks, widely used in object-oriented business software applications typically running in datacenter environments. We performed an empirical experiment aimed at determining the trade-offs in terms of performance and energy consumption of two different ORM solutions, Propel and TinyQueries, with respect to using plain SQL queries in the source code. Our experiment was planned using the Goal-Question-Metrics (GQM) approach [2], and the research question (RQ) driving our experimentation can hence be stated as follows: "What is the impact of the ORM frameworks on performance and power consumption, with respect to plain SQL?" Our experiment will result in concrete guidelines to developers and architects concerned with the energy efficiency of their applications, in particular when dealing with SQL-based databases.

This paper is structured as follows: in Section 2 we present

an overview of related work. In Section 3 we describe the context of our experiment (i.e. ORM approaches). In Section 4 we describe our experiment design in detail. In Section 5 we present the complete results of our experimentation. In Section 6 we reflect upon the results, providing additional insights and considerations. Section 8 concludes the paper.

2. RELATED WORK

Our approach to software energy efficiency research is based on an *inductive* approach [21], i.e. we build knowledge on software energy efficiency by gathering and analyzing empirical evidence. This is because the complexity of hardware-software interactions and multiple software layers affect energy consumption in a way that we are currently unable to deterministically describe e.g. using formal models. In a previous empirical study [21] we already found a significant impact of database-related practices on the energy efficiency of software applications. This triggered our interest towards this area, which appears promising, although scarcely investigated, in terms of potential optimizations.

The energy efficiency of database systems has been recently addressed in the scientific community [6, 9, 32], however most approaches focus on query optimization and scheduling [14, 3]. Schedulers can have a significant impact on performance (and consequently energy efficiency) for computing systems and thus they are suitable candidates for energy-efficient database research.

Other studies propose an energy cost model, using a variety of techniques and metrics, and integrate it into the query planners to direct it towards energy-efficient plans without sacrificing performance significantly [14, 3, 31, 33, 32, 13, 25, 27]. Others explicitly exploit energy saving features of hardware [13, 19, 20]. Novel approaches include maintaining an explicit banking of data-to-memory to shutdown memory banks [20, 19]. Few investigate algorithmic costs [19, 26, 1].

More generally, a significant number of studies published in 2014 and 2015 address the impact of best-practices to improve software energy efficiency. For example, Pinto et al. [18] analyzed and evaluated the impact in terms of energy consumption of 3 different thread management strategies, Li and Halfond [15] evaluate the impact of 3 best practices for Android application development, Linares-Vásquez et al. [16] aim at identifying whether some API calls are more energy-consuming than others, Sahin et al. [24], Perez-Castillo and Piattini [17] investigate the energy impact of commonly used refactorings. Most of these studies, however, focus on mobile applications. Obviously, in battery-powered and resource-constrained environments, energy concerns are linked to availability and functionality (i.e. battery life). In addition, measuring energy consumption on mobile devices is typically less complex than in large-scale IT environments.

To date, the impact of best practices for energy-efficient software in business and datacenter application scenarios still lacks significant empirical evidence. This study addresses this research gap: it studies the impact of a common practice in business applications using a commercial database and widely used technologies. As a representative datacenter-like environment, we used the Green Lab[1] [23] of the Vrije Universiteit Amsterdam. The Green Lab is a dedicated research lab to experiment on commercial software applications in a realistic environment and gather evidence on the energy efficiency of large-scale software applications. Currently, the lab is equipped with a rack of servers, instrumented with fine-grained power meters and sensors [4] to measure energy consumption at the level of single hardware components, if needed. All the datasets of the experiments conducted in the Green Lab are publicly available on its website, including those related to this study which can be found at the following URL: `http://goo.gl/0yZ9zB`.

3. EXPERIMENT CONTEXT

This case study regards object-relational mapping (ORM) frameworks. Such frameworks are developed in order to convert relational database entities (e.g. tables and records) to classes and objects to be used in object-oriented programming (OOP). This is because SQL is not structured to be easily integrated into OOP languages, often requiring the programmer to extensively use string concatenation for the creation of queries. Moreover, its syntax tends to become very complex, making it hard to read and maintain the code. Prominent examples of ORMs are Hibernate[2] for the Java language, Propel[3] for PHP and Sequel[4] for Ruby. Even if ORMs introduce several benefits to manage data of a RDBMS, compared to a plain SQL approach, these techniques have several disadvantages. A problem that characterizes many ORMs is the creation of many unnecessary objects that significantly decrease the overall performance of the queries. Another downside of ORMs is that, in order to reduce complexity, they hide many functionalities and features available in SQL.

In this case study, we performed an experiment using TinyQueries[5], provided by Diesveld Query Technology. TinyQueries is a commercial framework intended to solve the most common problems mentioned above, by keeping the performance and expressiveness of SQL while maintaining the simplicity of OOP. One of the basic ideas behind TinyQueries is to let the database do as much work as possible. There is reason to believe that databases can perform certain operations faster than modern programming languages: a database runs on compiled code and all vendors of RDBMS have put much effort in optimizing the execution time of queries. This dates back to the early age of computing, where the efficiency of algorithms was a high priority due to scarcity of resources. All these optimizations are still preserved in current RDBMS and still being improved. Performing the same operations of RDBMS in non-compiled code like Java, Ruby or PHP, generally decreases performance.

TinyQueries can be defined as a "no-ORM" framework: although it has some properties of an ORM, it is different from regular ORM in the sense that it does not map relational data to objects. To illustrate this with a simple example consider the case in which a developer wants to create a list of users and show their full names (first name and last name concatenated). Typically the first name and

[1] `http://www.s2group.cs.vu.nl/green-lab/`, last visited on June 23, 2016

[2] `http://hibernate.org/orm/`, last visited on June 23, 2016

[3] `http://propelorm.org`, last visited on June 23, 2016

[4] `http://sequel.jeremyevans.net`, last visited on June 23, 2016

[5] `http://www.tinyqueries.com`, last visited on June 23, 2016

		DB Framework		
		SQL	Propel	TinyQueries
Query Type	Create	*Medium*	*Small*	*Medium*
	Read	*Small*	*Large*	*Large*
	Update	*Medium*	*Medium*	*Large*
	Delete	*Medium*	*Medium*	*Small*

(a) Trial Set 1: Frameworks vs. Query Type

		DB Framework		
		SQL	Propel	TinyQueries
Table Size	Small	*Create*	*Read*	*Delete*
	Medium	*Read*	*Delete*	*Delete*
	Large	*Update*	*Read*	*Create*

(b) Trial Set 2: Frameworks vs. Table Size

Table 1: The two trial sets for the experiment.

the last name are stored in the database as separate fields. To get the list with the full names you can either choose to make a loop in Java or PHP that runs through the result-set and does the concatenation one by one, or you can make a SQL query that does the concatenation inside the query. In TinyQueries you do not have to write an SQL query, but you can define the concept "full_name" such that it is executed in the query itself. This is different from ORM in which you typically define a method "full_name" in the User-class, which causes it to be executed by the Java or PHP engine.

The TinyQueries framework has been analyzed in order to assess whether its adoption can increase the energy efficiency of software applications.

4. EXPERIMENT DESIGN

4.1 Experiment Definition

According to the Goal-Question-Metrics (GQM) approach [2] the definition of the goal of the experiment reported in this paper can be summarized as follows:

*"Analyze **different ORM frameworks** for the purpose of **evaluation** with respect to their **energy efficiency** as seen from the viewpoint of the **software engineer**, in the context of **database management**."*

We select the viewpoint of a software engineer because we want to establish best practices and recommendations for energy efficiency in data management. This is of relevance for software engineers to support the process of decision-making during the design phase and to optimize data access routines during development or refactoring.

For achieving the goal introduced in the previous section, we formulate the following **Research Question (RQ)**: *What is the impact of the ORM frameworks on performance and power consumption, with respect to plain SQL?*

For answering the question, the following **metrics** have been chosen:

- *Power Consumption:* the power consumed by the software system taken into consideration, measured in W.
- *Execution Time:* the execution time of an experimental unit, measured in seconds.

These metrics also represent the **dependent** variables (i.e. the observed quantities) of this study. As a derived metric, we will also provide results on energy consumption, calculated as $E = P \cdot \Delta t$, where P represents the average power consumption measured over Δt.

4.2 Variable Selection

Dependent variables have been already mentioned in the previous paragraph. In terms of **independent** variables (i.e. the quantities that can be manipulated and controlled), besides the obvious variable related to the ORM frameworks, we also identify two potential confounding factors, namely the *type* of query issued to the database and the *size of the table* interested by the query. Hence, we decide to explicitly control these variables, by selecting them as factors. Hence, the selected factors and treatments are the following:

- **Database framework**: two different database frameworks were analyzed: **Propel** and **TinyQueries**, in comparison with plain **SQL**.
- **Query Type**: the four basic functions of persistent storage were used in order to generate the measurements: **Create**, **Read**, **Update**, **Delete**.
- **Table size**: the size (in bytes) of the tables on which the previously listed queries operate. Three different sizes were taken into consideration as independent variables[6]. The table size is hereby defined as follows:
  - **Small table**: <100 KB
  - **Medium table**: >= 100KB and <1MB.
  - **Big table**: table >1MB.

4.3 Design type

For the identified variables, we adopted a design made of two distinct trial sets. Each set of trials was generated by varying two factors while randomizing the third, making use of a "Latin Square" design [30]. This way, the number of test cases to be analyzed remains relatively low if compared to the total number of interactions possible between the different treatments, making it feasible to carry out all the planned tests.

The two sets of trials are shown in Table 1. The assignment of samples to each trial was randomized.

4.4 Sample Selection

We drew our samples from the relational database of the online travel guide *favoroute*[7]. It consists of 28 tables, with 2,675,414 rows in total. The size of the database amounts to 371 MB. The samples taken into account for the experiment are mapped to the single trials in Table 1. The sampling method adopted for selecting the samples was convenience sampling. In fact, a selection that takes into consideration

[6]The ranges of the sizes were selected according to the files available in the analyzed database.

[7]`https://www.favoroute.com`, last visited on June 23, 2016

the structure and characteristics of the possible samples was deemed far more appropriate than a random selection of samples from the entire population. We selected the following tables:

- *googletype:* small table with 96 rows, 3 columns and 48,0 KB size
- *agenda:* medium table with 3,849 rows, 12 columns and 928,0 KB size
- *resources:* large table with 828,931 rows, 5 columns and 119,2MB size

Notice that the given row numbers and sizes denote the initial size before the start of the experiment, as some test cases include creation and subsequent deletion of rows.

4.5 Instrumentation

The experiment was conducted during April and May 2015 at the Green Lab of the Vrije Universiteit Amsterdam. All experiments described in this paper were carried out on a server equipped with two Quad-core Xeon processors, 18GB RAM memory, a 72GB HP SAS Drive with RAID 0 configuration and a P400i Smart Array RAID controller. The server runs Ubuntu 12.04 server edition, 64-bit. Measurements of power consumption are performed live by means of a Wattsup PRO meter[8], which measures power consumption of the whole system at a frequency of 1Hz (1 sample per second). Measurements of execution time are gathered by code instrumentation: the PHP script logs timestamps at the beginning and end of each experimental run.

4.6 Experiment Execution

To avoid bias from other processes running on the same machine, several baseline measures were done prior to experimentation and the mean of these baseline measures were taken into consideration before running the experiments. Execution time data was collected in nanoseconds, even though power consumption is measured on a lower frequency. This is to avoid interpretation problems due to variable rounding. Since a standard query typically takes significantly less than a second, queries were executed in batches of 1000 times in a row and an average power consumption per query was measured. Per each test script, the connection to the database was restarted, and new objects were created (in case of ORM frameworks). Caching of objects and queries was disabled.

Due to time constraints, we were able to perform only one trial per experimental unit (i.e. per each cell of Table 1).

4.7 Hypothesis Formulation

Our hypotheses are formulated in a two-tailed fashion. An ANalysis Of Variance (ANOVA) is conducted in order to perform hypothesis testing with significance level $\alpha = 0.05$.

In the hypotheses below, μ_p denotes the population mean power consumption, while μ_t denotes the population mean execution time.

4.7.1 Hypothesis 1: Query Types Power Consumption

Null hypothesis $H1_0$: There is no significant difference in power consumption between the different query types.
$\mu_{p,create} = \mu_{p,read} = \mu_{p,update} = \mu_{p,delete}$

Alternative hypothesis $H1_a$: There is a significant difference in power consumption between at least two of the different query types.
$\exists\, i,j \; with \; i \neq j \wedge i,j = Create, Read, Update, Delete|$
$\mu_{p,i} \neq \mu_{p,j}$

4.7.2 Hypothesis 2: Framework Execution Time

Null hypothesis $H2_0$: The execution time of the frameworks does not differ significantly from a query in plain SQL.
$\mu_{t,Propel} = \mu_{t,SQL} = \mu_{t,TinyQueries}$

Alternative hypothesis $H2_a$: Ho: The execution time of at least one of the two frameworks significantly differs from a query in plain SQL.
$\exists\, i,j \; with \; i \neq j \wedge i,j = Propel, TinyQueries, SQL|$
$\mu_{t,i} \neq \mu_{t,j}$

4.7.3 Hypothesis 3: Framework Power Consumption

Null hypothesis $H3_0$: The power consumption of the frameworks does not differ significantly from a query in plain SQL.
$\mu_{p,Propel} = \mu_{p,SQL} = \mu_{p,TinyQueries}$

Alternative hypothesis $H3_a$: The power consumption of at least one of the two frameworks significantly differs from a query in plain SQL.
$\exists\, i,j \; with \; i \neq j \wedge i,j = Propel, TinyQueries, SQL|$
$\mu_{p,i} = \mu_{p,j}$

4.7.4 Hypothesis 4: Table size

Null hypothesis $H4_0$: The size of the queried table does not have a significant effect on power consumption.
$\mu_{p,Small} = \mu_{p,Medium} = \mu_{p,Large}$

Alternative hypothesis $H4_a$: The size of the queried tables has a significant effect on power consumption.
$\exists\, i,j \; with \; i \neq j \wedge i,j = Small, Medium, Large|$
$\mu_{p,i} = \mu_{p,j}$

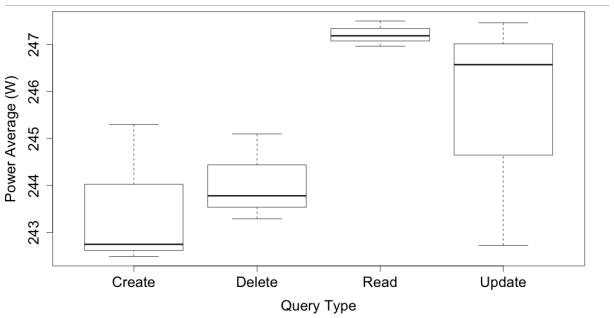
5. RESULTS

In this section, we present our results. First, we summarize the characteristics of our dataset using descriptive statistics. We also provide information about the distribution of our response variables. Afterwards, the formulated hypotheses are tested in order to assess if the *null* hypotheses can be rejected or not. In order to ease results interpretation, we present a graphical representation of the data distribution, by means of boxplots.

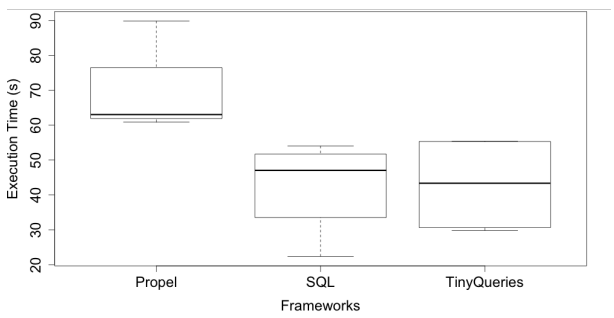
5.1 Descriptive Statistics

In Table 2, we present the descriptive statistics of the full dataset for our response variables (power consumption, execution time) and energy consumption. These statistics have been extracted by joining the two datasets mentioned in Section 4.3. The selected descriptive statistics are **mean**,

[8] `https://www.wattsupmeters.com/secure/index.php`, last visited on June 23, 2016



(a) Power consumption in Watts per query (H1)



(b) Execution time in seconds per framework (H2)

Figure 1: Box plots of results for hypotheses 1 and 2. NOTE: the Y Axis does not start from 0.

median, **standard deviation** (σ) and **coefficient of variation (CV)**.

As can be noticed from the values in Table 2, power consumption varied slightly during our experiment. The main variations occurred in terms of execution time, hence the effect on energy can be largely explained by the performance impact of the different treatments.

We also performed a Shapiro-Wilk normality test on our data, which failed to reject the null hypothesis for power consumption values (Trial Set 1: *W=0.8684, p=0.06239*; Trial Set 2: *W=0.9359, p=0.5394*) and execution time (Trial Set 1: *W=0.9446, p=0.5601*; Trial Set 2: *W=0.9252, p=0.4371*). We then assume the data is drawn from a normal distribution, which allows us to use the parametric Analysis of Variance for our hypothesis testing.

5.2 Hypothesis 1 results: Query types power consumption

For this hypothesis the overall means of the power consumption of the different query types were assessed. The used framework type was not considered in this analysis. The One-Way Analysis of Variance did **not** identify a significant difference for query types *(F(3,8)= 3.434, p=0.0724)*. Hence, **we reject the null hypothesis.** Figure 1a shows a boxplot of the distribution of the samples, grouped by query type.

The group means for each treatment have the following values:
CREATE: 243.51 W, DELETE: 244.05 W, READ: 247.22 W, UPDATE: 245.58 W.

Variable	Mean	Median	σ	CV
Power Consumption (W)	244.97	245.10	2.044	0.008
Execution Time (s)	51.95	55.00	17.58	0.34
Energy Consumption (Wh)	3.53	3.72	1.20	0.34

Table 2: Descriptive statistics of our complete dataset.

5.3 Hypothesis 2 results: Framework Execution time

This hypothesis was formulated in order to assess the difference between the selected frameworks *SQL*, *Propel* and *TinyQueries* in terms of execution time. The One-Way Analysis of Variance identified a significant difference for frameworks *(F(2,9)= 4.717, p=0.0397)*. Hence, **we reject the null hypothesis.** Figure 1b shows a boxplot of the distribution of the samples, grouped by framework.

The group means for each treatment have the following values:
SQL: 41.2 s; TinyQueries: 45.28 s (+9% wrt to SQL), Propel: 69.38 s (+68% wrt to SQL).

5.4 Hypothesis 3 results: Frameworks power consumption

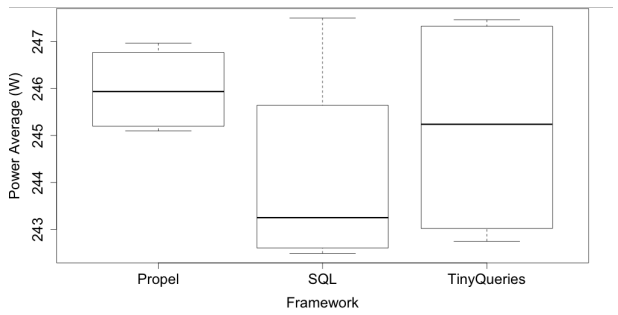
This hypothesis aims to assess the difference between the selected frameworks *SQL*, *Propel* and *TinyQueries* in terms of power consumption. The One-Way Analysis of Variance did **not** identify a significant difference for frameworks *(F(2,9)= 2.283, p=0.319)*. Hence, **we cannot reject the null hypothesis.** Figure 2a shows a boxplot of the distribution of the samples, grouped by framework.

The group means for each treatment have the following values:
SQL: 244.1 W, TinyQueries: 244.77 W (+0.2% wrt to SQL), Propel: 246.05 (+0.8% wrt to SQL).

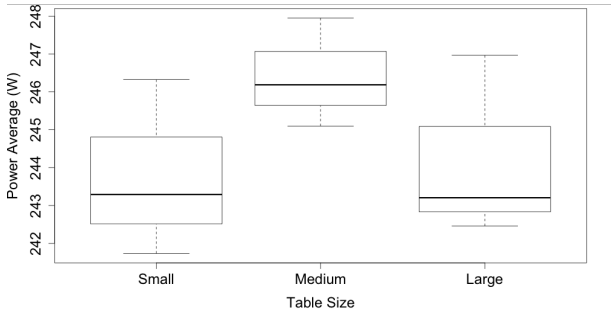
5.5 Hypothesis 4: Power consumption per table size

This hypothesis aims to assess the effect of table size (*Small, Medium, Large*) in terms of power consumption. The One-Way Analysis of Variance did **not** identified a significant difference for table sizes*(F(2,6)= 1.339, p=0.33)*. Hence, **we cannot reject the null hypothesis.** Figure **??** shows a boxplot of the distribution of the samples, grouped by table size.

The group means for each treatment have the following values:
Small: 244.57 W, Medium: 244.74 W (+0.06% wrt to Small), Large: 245.71 (+0.5% wrt to Small).



(a) Power consumption in Watts per framework (H3)



(b) Power consumption in Watts per table size (H4)

Figure 2: Box plots of results for hypotheses 3 and 4. NOTE: the Y Axis does not start from 0.

6. DISCUSSION

In addition to our hypothesis testing, we also performed a linear regression analysis on energy consumption values, using the complete dataset i.e. joining Trial Sets 1 and 2. We selected table size, framework and query type as predictors, which were checked for collinearity using generalized variance-inflation factors. Results are shown in Table 3, where per each predictor the regression coefficient and the standard error (in parenthesis) are reported. Units are in Watthours. The statistically significant coefficients are marked with a double or triple asterisk (**,***). Note that, as common practice in presence of qualitative predictors, factors have been decomposed in *dummy variables* i.e. true/false condition variables representing the occurrence of a specific level of the factor with respect to a baseline. For example, *FRAMEWORKSQL* indicates the impact of choosing SQL as a framework as compared to the baseline Propel, which does not appear in the table.

As can be seen from the table, both SQL and TinyQueries are shown to have a significant reducing effect on energy consumption, up to 2 Wh, with respect to Propel. We cannot conclude which one performs the best in such terms, as the difference between the two coefficients is within the margin of error. However, this is consistent with our hypothesis testing.

In order to identify the most energy–efficient framework, we performed post-hoc tests (namely, Tukey's Honest Significance Difference test [28]) on the impact of frameworks over energy consumption values. Results of the test are reported in Table 4 and the confidence intervals are represented in Figure 3. From these results, Propel appears significantly less energy efficient than both SQL and TinyQueries. This is most likely caused by the clear effect on performance of the three frameworks, as shown in Figure 1b. Propel is significantly slower than the other two approaches. This result was expected since TinyQueries and Propel are both based on *SQL*. Furthermore, Propel operates on objects instead of relying only on database tables. It is interesting to notice that the TinyQueries framework, created to keep a balance between high performance and OOP simplicity, is not significantly different, in terms of execution time, from plain SQL. On the contrary, the Propel framework resulted to be significantly slower than the others.

	Dependent variable:
	Energy
TABLEMedium	−0.296 (0.355)
TABLELarge	0.215 (0.433)
FRAMEWORKSQL	-1.940^{***} (0.356)
FRAMEWORKTinyQueries	-2.114^{***} (0.372)
ACTIONDelete	0.499 (0.410)
ACTIONRead	-1.206^{**} (0.421)
ACTIONUpdate	−0.578 (0.457)
Constant	5.220^{***} (0.413)
R^2	0.818
Adjusted R^2	0.720
Residual Std. Error	0.633 (df = 13)
F Statistic	8.342^{***} (df = 7; 13)
Note:	*p<0.1; **p<0.05; ***p<0.01

Table 3: Results of linear regression on energy consumption.

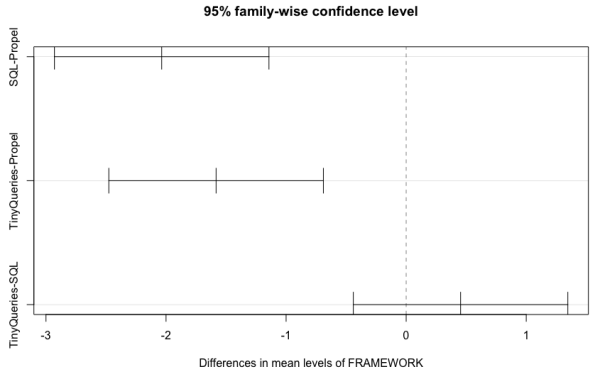


Figure 3: Difference in mean levels for different frameworks type.

Framework	Diff.	Lower Bound	Upper Bound	p-val. (adj.)
SQL-Propel	-2.036	-2.930	-1.143	0.001
TinyQueries-Propel	-1.583	-2.476	-0.690	0.001
TinyQueries-SQL	0.453	-0.440	1.346	0.399

Table 4: Results of Tukey's HSD test with 95% Confidence Intervals.

A more detailed analysis on the impact of both query types and table size provides more discussion points. In our regression model, we identify a significant effect of one particular query type: specifically, READ queries appear to consume about 1.2 Wh less than CREATE queries which have been selected as a baseline. This is consistent with the boxplot in Figure 1a, which shows READ queries consume more on average. This might be due to a different resource usage pattern: READ queries perform might have less I/O activity than other type of queries, hence resulting in a higher CPU usage which in turn causes higher power consumption. This does not necessarily contradict our hypothesis testing, as this effect cannot be generalized for all the types of queries we evaluated. However, it is likely that due to the low power of our analysis, we fail to reject the null hypothesis related to the power consumption of the different query types. In a previous empirical study [22] we showed that query optimizations can significantly affect power consumption. We will further investigate this effect in future research, by also collecting resource usage data.

As shown in Figure 4, Propel is particularly slow in executing *Delete* query types. This can be attributed to the complexity overhead in the frameworks storage procedures, for example due to unnecessarily created objects that subsequently have to be disposed of. Another interesting finding is that Propel spent more time when working on *Medium*-sized tables rather than *Large*-sized ones. This might be due to the fact that the *Medium*-sized table has more columns (12) than the *Large* one (5), hence the ORM framework has to elaborate more fields for each created object. This indicates that table width might be a more important factor than size, for determining the overall performance of an ORM framework.

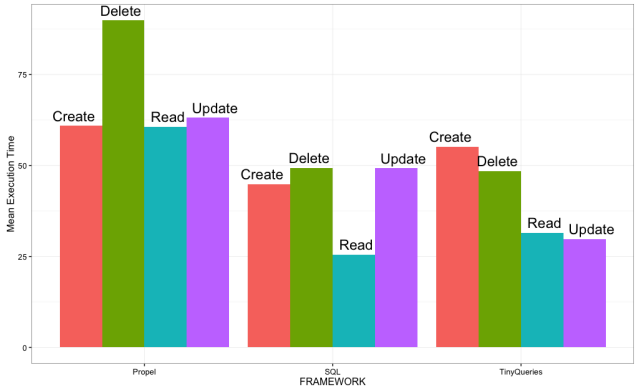


Figure 4: Bar plot of the execution time in seconds per framework and query type

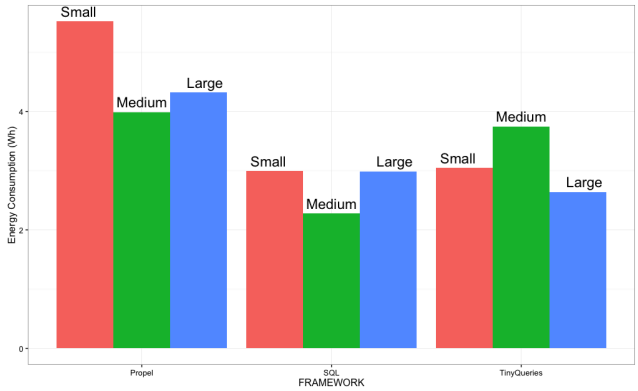


Figure 5: Bar plot of the energy consumption in Watthour per framework and table size

In terms of power consumption, plain SQL seems to consume less on average, while Propel seems to consume more than TinyQueries (see Figure 2a in Section 5). However, it is worth noticing that no statistically significant effect of the frameworks with respect to average power consumption was identified. Thus, execution time is the key factor that determines software energy efficiency in this case: SQL clearly consumes less energy, as it spends less time to perform the same number of queries, while Propel is the least energy efficient (see Figure 5). This is a clear situation where performance and energy efficiency are positively correlated.

A clear impact of ORM frameworks in terms of performance and energy efficiency has emerged. This is a valuable contribution for software engineering, as it introduces the problem of choosing between the higher maintainability and flexibility that ORM approaches provide and the lower energy costs and execution time of plain SQL approaches. A various number of factors could influence this decision: budget, time constraints, performance, staff practical knowledge, and many others.

TinyQueries could be a suitable solution to handling explicit SQL queries, without the performance overhead of a full ORM framework. It provides a similarly simple in–code handling of database transaction as a full–fledged framework such as Propel does, while maintaining a performance very close to that of native SQL queries.

7. THREATS TO VALIDITY

In the following subsections we identify the main threats to validity for the experiment. This section follows the structure indicated by Wohlin et al. [30] classifying the threats according to validity types.

7.1 Construct validity

The measurements gathered for the reported experiment were energy consumption and execution time of database queries. These two metrics were measured live, therefore some heuristics were used in order to reduce the noise present in the measurement results as much as possible. In order to measure the power consumption of the queries presented in Section 4.3, several baseline measures were carried out before the experiment measurements took place. The mean of these baseline measures was considered in the analysis of

the gathered data as a control factor in order to correctly evaluate the results.

7.2 Internal validity

The threats to internal validity have been mitigated by introducing two different experimental designs, as described in Section 4.3. In each of these experimental designs, a factor has been randomized in order to isolate the effects of the main factors as much as possible.

Apart from this, the interaction effect between the different factors has also been taken into account by carrying out an analysis of variance on the gathered data sets. By using these approaches, the possible internal validity threats of the experiment of this research have been reduced as much as possible, limiting the possibility of unaccounted factors influencing the measurements to an acceptable level. During the experiment the researchers kept also track of the environment in order to assess if any other contaminating factors, such as delay or temperature, were influencing the experiment.

Another possible threat to internal validity is the use of the same samples multiple times to assess the influence of different factors and treatments. The usage of experimental objects multiple times for different measurements could highly influence the results, as a measurement carried out previously could potentially alter the analyzed sample and therefore influence other measurements. In the population taken into consideration this threat is not present, as once a treatment is applied, the experimental objects can be restored to their initial state, i.e. the state of the sample before the treatment was applied. Moreover some countermeasures, as for example avoiding the use of caching in order to execute a fair comparison between the ORM systems, where adopted.

Another possible threat to the internal validity is the different measurement time intervals used by the tools in the experiment. While the execution time of the analyzed queries is measured in nanoseconds, the power-consumption measurements, carried out with a *Watts Up Pro* power meter, are measured in seconds. In order to avoid the introduction of inaccuracies due to rounding approximations into the gathered data set, each trial is composed by 1000 query executions, as mentioned in Section 4. This way, the execution time of a batch of identical queries was measurable in seconds and could therefore be mapped to the power-consumption measurements. Even though these heuristics made the comparison of the different metrics possible, some small inaccuracies, in the order of the milliseconds, could not be avoided.

Although the developer of TinyQueries is one of the authors of this paper, but he did not take part in the experimentation described in this study. His role was to provide data and technical expertise for the implementation phase, as well as insights and advise on the interpretation of the results. This did not affect in any way the validity of the results, as the experiment was carried on independently by a separate team of researchers, hence avoiding potential experimenter bias.

7.3 Conclusion validity

A possible threat to conclusion validity is our small sample size. Due to time constraints, we were able to perform only one trial per experimental unit. This led to a quite low number of samples, hence reducing the power of our analysis. However, we mitigated this threat by duplicating the test sets and randomizing the treatment assignment. In addition, we verified the normality assumption for the ANOVA test, which is anyway proved to be quite robust even in cases of slightly non-normal datasets. This ensures that we always select the test with the best power with respect to its assumptions. Given these considerations, we are performing an exploratory analysis where we are interested in major differences and not in precise values. Hence, a low power (i.e. a higher risk of type II errors) is not a primary concern for our study.

7.4 External validity

A potential threat to external validity for the experiment under consideration is due to the sampling method used to select the samples. In fact the samples were drawn from the population by choosing them based on their availability and convenience, leading to the adoption of a convenience sampling approach.

Moreover only three different table samples, reported in Section 4.3, were used as experimental objects. This decision could potentially lead to results that are specifically bound to the samples selected for the experiment. It is therefore important to consider the possibility that a generalization of the results to the entire population could not be possible as the chosen samples might be not representative enough. However, the selected samples that were all drawn from the *Favoroute* database, a real database used in production and thus representative of the majority of the population. These samples where chosen by identifying the characteristic traits present in the population, e.g. size of the database tables, and by ensuring that all these representative samples were included in the experimentation.

Our raw data, scripts and plots have been made available as an online package, to ease reproduction and replication, at the following URL: `http://goo.gl/0yZ9zB`.

8. CONCLUSIONS

In this paper, we presented an experiment aimed at determining the trade-offs in terms of performance and energy consumption of different ORM solutions. Our RQ was stated as follows: *What is the impact of ORM frameworks on performance and power consumption, with respect to plain SQL?*

The experiment results shows that ORM frameworks have a significant impact in terms of both performance and energy consumption with respect to plain SQL. Pure ORM approaches such as Propel can introduce a 70% increase in execution time, significantly increasing energy consumption. Hence, this creates a trade–off between the benefits in maintainability and flexibility granted by ORM approaches and the loss in energy efficiency. The results also show how a "no-ORM" framework, such as TinyQueries, can provide a better balance between the two aspects, assuming it provides the same benefits of ORM in terms of maintainability.

In conclusion, we can summarize our results in the following *guideline* for energy–efficient data management: *to achieve the best energy efficiency and performance of an application using a SQL database, implement the queries in plain SQL or adopt a "no-ORM" approach, such as TinyQueries.*

This guideline will complement our online library of best-

practices for energy-efficient software development[9] that serves as public repository for our research efforts in software energy efficiency. Our future research work will further evaluate the impact of the best practices already available, with the ultimate goal of delivering a reliable body of knowledge on software energy efficiency.

Acknowledgements

This work has been supported by the RAAK-MKB innovation fund from SIA, the Dutch National Taskforce for Applied Research, as part of the Greening the Cloud project and by the Netherlands Enterprise Agency (RVO) as part of the GreenServe project. We want to thank our students Mylene van der Koogh, Roberto Verdecchia and Nina Wolfram for their work during the Green Lab course.

9. REFERENCES

[1] N. An, S. Gurumurthi, A. Sivasubramaniam, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin. Energy-performance trade-offs for spatial access methods on memory-resident data. *The International Journal on Very Large Data Bases*, 11(3):179–197, 2002.

[2] V. Basili and G. Caldiera. Rombach. HD the goal question metric approach. *Encyclopedia of Software Engineering, Wiley*, 2(1994):528–532, 1994.

[3] C. Chen, B. He, X. Tang, C. Chen, and Y. Liu. Green Databases Through Integration of Renewable Energy. In *6th Biennial Conference on Innovative Data Systems Research (CIDR)*, 2013.

[4] M. Ferreira, E. Hoekstra, B. Merkus, B. Visser, and J. Visser. Seflab: A lab for measuring software energy footprints. In *Green and Sustainable Software (GREENS), 2013 2nd International Workshop on*, pages 30–37, May 2013.

[5] E. Gelenbe and Y. Caseau. The impact of information technology on energy consumption and carbon emissions. *Ubiquity*, 2015(June):1:1–1:15, June 2015.

[6] S. Götz, T. Ilsche, J. Cardoso, J. Spillner, T. Kissinger, U. Aβmann, W. Lehner, W. E. Nagel, and A. Schill. Energy-efficient databases using sweet spot frequencies. In *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pages 871–876. IEEE Computer Society, 2014.

[7] C. Gough, I. Steiner, and W. Saunders. *Energy Efficient Servers: Blueprints for Data Center Optimization*, chapter Data Center Management, pages 307–318. Apress, Berkeley, CA, 2015.

[8] A. Hindle. Green mining: Investigating power consumption across versions. In *ICSE*, pages 1301–1304, jun 2012.

[9] N. Iimura, N. Nishikawa, M. Nakano, and M. Oguchi. A proposal of storage power control method with data placement in an environment using many hdds. In *Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication*, IMCOM '15, pages 73:1–73:8, New York, NY, USA, 2015. ACM.

[9] `https://wiki.cs.vu.nl/green_software/index.php/Best_practices_for_energy_efficient_software`, last visited on June 23, 2016

[10] J. G. Koomey, S. Berard, M. Sanchez, and H. Wong. Implications of historical trends in the electrical efficiency of computing. *IEEE Ann. Hist. Comput.*, 33(3):46–54, Mar. 2011.

[11] P. Lago, R. Kazman, N. Meyer, M. Morisio, H. A. Müller, F. Paulisch, G. Scanniello, B. Penzenstadler, and O. Zimmermann. Exploring initial challenges for green software engineering: summary of the first GREENS workshop, at ICSE 2012. *SIGSOFT Softw. Eng. Notes*, 38(1):31–33, Jan. 2013.

[12] P. Lago, N. Meyer, M. Morisio, H. Müller, and others. Leveraging energy efficiency to software users: summary of the second GREENS workshop, at ICSE 2013. *ACM SIGSOFT Software*, 2014.

[13] W. Lang, R. Kandhan, and J. M. Patel. Rethinking Query Processing for Energy Efficiency: Slowing Down to Win the Race. *IEEE Data Eng. Bull.*, 34(1):12–23, 2011.

[14] W. Lang and J. M. Patel. Towards eco-friendly database management systems. In *4th Biennial Conference on Innovative Data Systems Research (CIDR)*. www.cidrdb.org, 2009.

[15] D. Li and W. G. J. Halfond. An investigation into energy-saving programming practices for android smartphone app development. In *Proceedings of the 3rd International Workshop on Green and Sustainable Software*, GREENS 2014, pages 46–53, New York, NY, USA, 2014. ACM.

[16] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk. Mining energy-greedy api usage patterns in android apps: An empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 2–11, New York, NY, USA, 2014. ACM.

[17] R. Perez-Castillo and M. Piattini. Analyzing the harmful effect of god class refactoring on power consumption. *Software, IEEE*, 31(3):48–54, May 2014.

[18] G. Pinto, F. Castor, and Y. D. Liu. Understanding energy behaviors of thread management constructs. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA '14, pages 345–360, New York, NY, USA, 2014. ACM.

[19] J. Pisharath, A. Choudhary, and M. Kandemir. Energy Management Schemes for Memory-resident Database Systems. In *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management*, CIKM '04, pages 218–227, New York, NY, USA, 2004. ACM.

[20] J. Pisharath, A. Choudhary, and M. Kandemir. Reducing Energy Consumption of Queries in Memory-resident Database Systems. In *Proceedings of the 2004 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '04, pages 35–45, New York, NY, USA, 2004. ACM.

[21] G. Procaccianti, H. Fernandez, and P. Lago. Empirical evaluation of two best practices for energy-efficient software development. *Journal of Systems and Software*, 2016. DOI: 10.1016/j.jss.2016.02.035.

[22] G. Procaccianti, H. Fernandez, and P. Lago. Empirical evaluation of two Best-Practices for Energy-Efficient

software development. *J. Syst. Softw.*, 117(July 2016):185–198, 2016.

[23] G. Procaccianti, P. Lago, A. Vetro, D. Méndez Fernández, and R. J. Wieringa. The green lab: Experimentation in software energy efficiency. In *International Conference on Software Engineering (ICSE)*, 2015.

[24] C. Sahin, L. Pollock, and J. Clause. How do code refactorings affect energy usage? In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '14, pages 36:1–36:10, New York, NY, USA, 2014. ACM.

[25] J. Song, T. Li, X. Liu, and Z. Zhu. Comparing and analyzing the energy efficiency of cloud database and parallel database. In *Advances in Computer Science, Engineering & Applications*, pages 989–997. Springer, 2012.

[26] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah. Analyzing the Energy Efficiency of a Database Server. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 231–242, New York, NY, USA, 2010. ACM.

[27] Y.-C. Tu, X. Wang, B. Zeng, and Z. Xu. A system for energy-efficient data management. *ACM SIGMOD Record*, 43(1):21–26, 2014.

[28] J. W. Tukey. Comparing individual means in the analysis of variance. *Biometrics*, 5(2):99–114, June 1949.

[29] N. Wirth. A plea for lean software. *IEEE Computer*, 28(2):64–68, Feb. 1995.

[30] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[31] Z. Xu. Building a Power-aware Database Management System. In *Proceedings of the Fourth SIGMOD PhD Workshop on Innovative Database Research*, IDAR '10, pages 1–6, New York, NY, USA, 2010. ACM.

[32] Z. Xu, Y. Tu, and X. Wang. Online Energy Estimation of Relational Operations in Database Systems. *Computers, IEEE Transactions on*, 64(11):3223–3236, Nov. 2015.

[33] Z. Xu, Y.-C. Tu, and X. Wang. Exploring power-performance tradeoffs in database systems. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 485–496, Mar. 2010.