EPiC Series in Computing

Volume 52, 2018, Pages 1–19

ICT4S2018. 5th International Conference on Information and Communication Technology for Sustainability

# An Empirical Evaluation of Database Software Features on Energy Consumption

Sedef Akınlı Koçak[1], Gulfem Işıklar Alptekin[2], Andriy Miranskyy[1], Ayşe Başar Bener[1], and Enzo Cialini[3]

[1] Ryerson University, Toronto, Ontario, Canada
`sedef.akinlikocak@ryerson.ca, avm@ryerson.ca, ayse.bener@ryerson.ca`
[2] Galatasaray University, Istanbul, Turkey
`gisiklar@gsu.edu.tr`
[3] IBM Toronto Laboratory, Toronto, Ontario, Canada
`ecialini@ca.ibm.com`

**Abstract**

Although software does not consume energy by itself, its characteristics determine which hardware resources are made available and how much energy is used. Therefore, energy efficiency of software products has become a popular agenda for both industry and academia in recent years. Designing such software is now a core initiative of software development companies aiming toward social responsibility. Meanwhile, however, developing environmentally sustainable software products is a challenge in that performance, functionality and energy consumption can reflect conflicting goals. In this paper, our objective is to analyze the effects of different features on energy consumption of the IBM DB2, a commonly used database product. The empirical work focuses on three features. We executed a workload in preconfigured software with some features enabled or disabled and with different numbers of users. To compare the different scenarios, three sets of green metrics were utilized. The metric set identified various parts of the software system where energy is consumed. Our findings may suggest that the conflicts among software system performance, functionality, and energy consumption can be mitigated by choosing a combination of features that interact in a way that improves energy efficiency. *Index Terms* energy consumption, green metrics, energy efficiency, environmental sustainability, software feature interaction, database.

## 1 Introduction

Increasing usage of information and communication technologies (ICT) is continuously impacting overall energy consumption, particularly in view of rising energy costs. Green IT/ICT has been defined with the goal of designing, manufacturing, using, and disposing of computers, servers, and associated subsystems efficiently and effectively with minimal, or no, impact on the environmen [27]. Although, there have been many researchers who work on power consumption and monitoring of embedded systems [32], the energy consumption of software has recently been gaining the attention of research community [19, 8]. Software engineering does not currently provide consolidated knowledge on the relationship between software products and their energy consumption [16, 25, 29, 26].

Software features are considered an expression of the users requirements [9] and an increment in product functionality [23]. According to Kang et al. [18], a feature may be defined as "a distinctively identifiable functional abstraction that must be implemented, tested, delivered, and maintained". The term "feature" and "functionality" are often used interchangeably. In this paper a feature is used as an implementation of a functionality. Software evolution can be described as the process of adding and removing these features [23]. Feature-based design facilitates such process. The idea of feature oriented software development (FOSD) is to decompose a software system in terms of the features it provides [2]. FOSD has two challenges: (1) consistently mapping the features to user needs and (2) the tendency of features to interact unintentionally [2]. Therefore, it is important to examine in detail the design and interactions of features in the context of energy behaviour of the software. These findings may guide the reasoning used by developers in combining features in a way that satisfies users, but also meets environmental standards.

To the best of our knowledge, no empirical study has defined the impact of feature interaction on energy consumption.

Data compression, one of the most frequently used DB2 features, reduces the number of I/O operations, while increasing CPU utilization. In our previous study [21], we examined in detail the impact of only data compression on software energy consumption. Results showed that energy consumption is inconsistent with the variability of workload. Ideally, the energy efficiency of the application should be proportional to the intensity of its workload. In practice, however, this is rarely seen in production systems [4]. In real programming scenarios, software developers create various interacting features. Thus, even though each feature individually works as expected, certain combinations of features may lead to unexpected behaviour.

In this paper, we concentrate on a more realistic software system configuration that uses three functional features: data compression, design advisor *(DA)* indexes *(I)*, and materialized query tables *(MQT)*. The experiments, consisting of six different scenarios, were conducted on software using the same workload and aimed toward examining the individual and cumulative effects of software feature interactions on the test system's energy efficiency. Cumulative effects are changes caused by the combined impact of the features. As noted above, individual features may work as expected, but the modification of an existing feature or addition of new features can trigger an unanticipated cumulative effect not anticipated in the original development plan.

Our research questions and their rationales are as follows:

*RQ1: How should software energy efficiency be evaluated?*

In order to provide users with energy efficiency along with other desired software features, it is important to evaluate the energy consumption of a product. Evaluation criteria should be applicable both to new and legacy systems. Building a new system may not always be feasible because of the economic and technical prohibitions. Evolution of the existing popular legacy systems must also be considered. Therefore, we defined a simple set of metrics to measure the energy efficiency of any given software as well as its functionality.

*RQ2: What are the individual/cumulative effects of software features on energy consumption?*

New features will continuously be added to a software system. A feature may, for example, be expressed as new user interfaces, new APIs, or new functionalities. However, introducing new features might also introduce some risks along with cost. One of the major risks is feature interaction which occurs when the behaviour of one feature is affected by the presence of another feature. This occurrence cannot be easily detected from the behaviour of the individual features involved [1]. Therefore, in order to develop well-structured software, as well as minimize costs, these interactions and their impacts on system behaviour should be carefully examined.

Three sets of metrics determined experimental outcomes: IT resource metrics (CPU usage, I/O usage, and storage space), lifecycle metric (application performance) and energy impact metrics (system energy usage and application energy efficiency). The results show that (1) different combinations of software features may create different resource usage behaviour and, hence, different levels of energy consumption, (2) the cumulative effect of all features is more significant than the effect of each individual feature, (3) feature interactions can be detected, but with no guarantee of feature-specific behavior in a group, and (4) common resource indicators, such as execution time and application performance can sufficiently detect the energy impact of software. Such results offer initial insights into the levels of feature-driven energy consumption, enabling software developers to more precisely determine the utility of a new software feature in the context of its interaction with other features and, hence, the overall impact on energy consumption before its implementation.

The remainder of this paper is organized as follows: Section 2 provides a review of related literature on energy management and metrics. Section 3 describes the methodology, including experimental setup and data collection. DB2 features are also given in Section 3. The results and related discussion are given in Section 4, before concluding the paper in Section 5.

## 2 Related Work

Most studies of green IT have focused on the direct environmental impact of hardware [12, 31, 13]. Researchers have argued that software is not a material product, and, therefore, it alone cannot consume energy [15]. Yet, software drives the hardware of a system. Accordingly, Hilty and Lohmann [14] emphasized that the process of software development plays a specific role in creating indirect effects on the environment. This is especially true when a particular kind of software is demanded or when software is developed within a tight schedule and budget.

Extensive research in the area of systems energy efficiency has been conducted in the IT domain. While non-profit organizations, such as the Transaction Processing Performance Council (TPC)[1] and the Standard Performance Evaluation Corporation (SPEC)[2], are defining energy benchmarks, researchers are also proposing frameworks and metrics [20, 24, 17]. An EU project focused on green IT in data centers based on a “Games” framework, as proposed by Kipp et al. [20]. They divided green metrics into four main classes: a) IT resource usage metrics, b) application lifecycle metrics, c) energy impact metrics, and d) organisational metrics. IT resource usage metrics consist of CPU usage, storage, memory usage, and I/O usage of an application. Application lifecycle metrics measure the cost of setting the parameters and monitoring green metrics. Energy impact metrics measure the impact of IT service centers and applications on the environment, considering power supply, consumed materials, emissions, and other energy-related factors. Organisational metrics measure the impact on infrastructural costs. Mahmoud and Ahmad [24] defined a framework to show the relationship between green computing and green performance metrics using the metrics defined by Kipp et al.[20], along with hardware and network metrics. All of the models, frameworks and benchmarks have various limitations, and thus, they are not more extensively presented here. Since most of them are proposed for data centers, they are not well suited to assess heterogeneous system types.

Although most research on IT energy efficiency is related to hardware, the role of software is increasingly driving energy consumption. For example, in the last few years, developers have

[1]http://www.tpc.org/tpc_energy/default.asp
[2]http://www.spec.org/benchmarks.html/power

leaned toward considering software technologies in terms of energy optimisation [6]. However, Naumann et al. [28] reported on the lack of software models in the area of environmental sustainability. To address this, they developed a reference "GreenSoft" model that defines green software metrics and green software design and development processes. Capra et al. [8] designed a metric that measures energy efficiency relative to functionally similar systems. Arnoldus et.al [3], followed by Kalaitzoglou et al. [17], employed the Goal-Question-Metric (GQM) methodology [5]. Arnoldus et al. [3] proposed three energy efficiency metrics, namely, annual consumption, average energy consumption per transaction and relative energy efficiency for e-services. Kalaitzoglou et al. [17] defined metrics based on energy behaviour, capacity, and resource utilisation. Mahmoud and Ahmad [24] stated that most studies do not clearly explain how to apply existing metrics and models. Procaccianti [29] analyzed the correlation between energy and software applications, looking for patterns and mechanisms that affect energy consumption. He found that the complexity of software and hardware interaction results in the interplay of many factors that impact energy consumption. Taken together, it appears that no "gold standard" green metrics able to assess energy efficiency in all contexts is in place.

## 3 Methodology

In this section, the experimental setup are explained in detail.

### 3.1 Metric Evaluation

Several internal factors, including type of systems or specific hardware resources used by executing the application, system configuration, functional suitability, and external factors, such as application performance, quality requirements, workload, and any other external constraints, affect the energy consumption of software [22]. Measuring only the activity of CPU, disk, and/or the I/O channels at a given point in time may not provide a good indication of the amount of real work that a system performs. It is also necessary to measure the actual completed unit of work during this period of time. To do this, we adopted a simple set of metrics, including IT resource metrics, lifecycle metrics and system energy usage metrics, as defined by Kipp et al. [20], and application energy efficiency metrics adopted from the TPC-Energy Primary Metric [11] (Table 1).

| **Adopted Metrics** | **Unit** |
|---|---|
| **IT Resource Usage Metrics** | |
| CPU usage rate | % |
| I/O usage | % |
| Storage usage | % |
| **Lifecycle Metrics** | |
| Application performance (AP) | # of useful unit of work/kWh |
| **Energy Impact Metrics** | |
| System energy usage (EC) | kWh |
| Application energy efficiency (AEE) | Wh# of transactions |

Table 1: Metrics used to evaluate energy efficiency

CPU usage is measured as a percentage of processor utilization when using an application to perform specific computing operations. I/O usage is determined by the percentage of utilization of the corresponding I/O device for communications and the number of messages transmitted by an application over a set of system components [20]. Application performance is measured as the amount of useful unit of work per kWh for a specific application type (Eq. 1). A useful unit of work ($UUW$) is defined as a measurable quantity of work done. It depends on the type of application. For example, if it is a server application, it is referring to number of statement completed. System energy usage (EC) is power consumption in unit of time (kWh).

For the purposes of this paper, it is also convenient to define the energy required to compute a single unit of work. Application energy efficiency ($AEE$) is measured in Watt-hours per transaction (or scaled to kWh per number of statements) and is calculated as Eq. 2.

$$AP = UUW/EC. \tag{1}$$

$$AEE = EC/\#\, of statement. \tag{2}$$

Storage usage *(SU)* is measured as a percentage of entire storage utilized on the corresponding storage device, as

$$SU = Used\, disk\ space/Allocated\, disk\ space. \tag{3}$$

The relationship between storage usage and space saving has the following expressions [30]:

*Space saving=1-(Compressed data size/Uncompressed data size)* or

*Space saving=1-(Actual data size/Raw data size).*

System energy usage implies energy efficiency, but it ignores system performance. On the other hand, application performance helps to compare different systems and configurations, but it is not adequate for the independent evaluation of energy savings or performance. Therefore, it is necessary to evaluate all these metrics simultaneously to determine the total energy impact of a given software application from different viewpoints (i.e., users, developers, and project managers).

## 3.2 Experimental Setup

Our software under study is IBM DB2 v.10.1 database system for Linux, UNIX and Windows. DB2 is commonly used relational database management software. The experiments aim to examine the individual and cumulative effects of different software features on the energy consumption of DB2. Users of modern database systems typically use these three DB2 features: DB2 Adaptive Data Compression (denoted by C), DB2 Design Advisor – Indexes (denoted by I), and DB2 Design Advisor – Indexes and Materialized Query Tables (denoted by I+MQT). Not all editions of DB2 have all of the functionality available. We run our test on the “DB2 Advanced Enterprise Server Edition” in which all the features under study are present. These features are discussed below.

### 3.2.1 DB2 Adaptive Data Compression (C)

Disk storage is often the most expensive component of a database solution. Even a small reduction of the storage system may result in substantial cost savings for the entire database solution.

### 3.2.2 DB2 Design Advisor: Indexes (I)

The DB2 Design Advisor (*DA*) is a tool that significantly improves workload performance [34]. The *DA* automatically analyzes workload queries. Then, based on query structure, the *DA* suggests which additional database objects can be created to speed up execution of the workload. One of the types of objects, which the *DA* can suggest, is a database index. An index is a data structure that speeds up the data retrieval from database tables. Moreover, to use these structures comes at the cost of additional storage for index data and CPU cycles to maintain and access an index. In our experiments, in some scenarios, we will add indexes, denoted as *I*, recommended by the *DA*.

### 3.2.3 DB2 Design Advisor: Indexes and Materialized Query Tables (I+MQT)

In addition to indexes, the *DA* can also recommend creating materialized query tables (*MQT*). The *MQT* can save the results of a query in a new table and refresh it, either dynamically or on demand, if data in the underlying tables are altered. *MQTs* can improve performance of complex and repetitive queries, sometimes by orders of magnitude, because *MQTs* have to be computed just once, thus avoiding re-computation of high-cost query operations, such as joins and sorts. In our experiments, in some scenarios, we will add *MQTs* and indexes[3], as denoted by *I+MQT*, suggested by the *DA*.

| | **DA disables** | **DA enabled with I** | **I+ MQT enabled** |
|---|---|---|---|
| **Compression disabled** | C- DA- (Scenario 1) | C- I+ (Scenario 2) | C- I+ MQT (Scenario 3) |
| **Compression enabled** | C+ DA- (Scenario 4) | C+ I+ (Scenario 5) | C+ I+ MQT (Scenario 6) |
| Queries: 240; Data size: 1GB; Number of users: 1-2-4-8 | | | |

Table 2: Experimental design parameters

For Scenario 1 (C-DA-), we did not enable compression or add any of the objects suggested by the DA. For Scenario 2 (C-I+), we enabled only the design advisor with indexes. For Scenario 3 (C-I+MQT), we disabled compression, but enabled design advisor with MQTs. For Scenario 4 (C+DA-), we enabled compression without adding any of the DA-recommended objects. For Scenario 5 (C+I+), we enabled compression and DA with indexes. For Scenario 6 (C+I+MQT), all the features were enabled. Our reference workload is online analytical processing (OLAP) type 5 from "TPC Benchmark H" (TPC-H) [10]. TPC-H is a decision support benchmark and is considered the standard benchmark in the database community.

For every scenario, we generated an empty database, created the required objects, and, in the case of Scenarios 4, 5, and 6 (containing C+ in their names), enabled the compression feature. We then populated the database with 1 GB of raw data. In order to reflect the impact of number of users on workload performance, we simulated concurrent execution of the queries, simultaneously connecting 1, 2, 4, or 8 users.

Before starting the experiments, we generated 1 GB of raw data and 240 distinct queries associated with these raw data. For every scenario and number of concurrently connected users, the 240 queries were executed sequentially for approximately two hours in a circular fashion. For every workload execution, we counted the number of statements executed in this two-hour

[3]The indexes may be associated with both regular tables and MQTs.

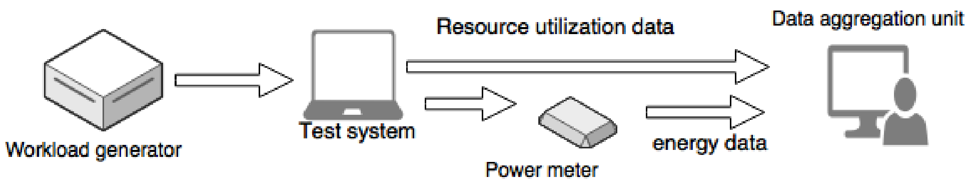


Figure 1: Testbed setup

time interval and measured the amount of CPU, I/O, and energy consumed by the computer. For every permutation of the configured scenarios and number of users connected, we executed the workload four times. The results of the first run differed significantly from the results of the remaining three runs because the autonomous functionality of DB2 needs some time to adjust default engine and database configuration to a given workload. Therefore, we discarded the measurement for the first run and used the average of the results of the remaining three runs in our analysis. We also measured CPU, I/O, and energy consumption of the computer in its idle state to establish a baseline.

## 3.3 Testbed setup and measuring system statistics

The computer used in our experiments has Intel Core i5-540M dual core 2.53 GHz CPU, 4GB of RAM, and 250GB of storage on a magnetic hard drive (HDD) Toshiba MK2529GSG with Ubuntu Linux OS v.12.04, Desktop edition. We disabled OS graphical user interface to minimize the number of background processes, making it "closer" to Server edition. The computer was dedicated to our workloads – no other tasks were executed on this machine concurrently.

Memory and disk allocation are reported by the DB2 itself. For measuring CPU and I/O load we used Sysstat "sar" command [4]. CPU utilization is computed as a 3-tuple of CPU utilization that occurred while executing at user level, user level with "nice" priority [5], and at the system level. The higher the number – the more utilized the CPU is. Note that our computer has two core CPU. Therefore, 50% utilization means full load of one core.

Transfers per second shows the number of I/O requests (of indeterminate size) to the hard drive. The higher the number the more data active in our reads and writes to/from the hard drive. The transfers per second data in this figure is computed by summing up per second data collected by Sysstat.

## 3.4 Energy Consumption Measurement Method

Figure 1 illustrates our testbed. The workload was generated by the workload generator and applied to the test system. The power meter, which was used to measure the energy consumption, is "UPM EM100–Energy Meter"[6]. It has a resolution of 0.01 kWh. We took the cumulative power consumption (kWh) measurement at the beginning and at the end of the workload execution. The meter, internally, samples power consumption and then integrates it.

[4]SYSSTAT manual. http://sebastien.godard.pagesperso-orange.fr/

[5]Unix scheduling based on a priority calculation where the nice value adjusts the priority

[6]http://cache-m2.smarthome.com/manuals/1139.pdf

| | **Storage usage (%)** | **Statement count ($x10^3$)** | | | | **Total CPU ($\%user + \%nice + \%system$)** | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | **1** | **2** | **4** | **8** | **1** | **2** | **4** | **8** |
| C- DA- | 0 | 1.18 | 8.08 | 11.36 | 7.32 | 10.1 | 62.9 | 84.5 | 33.8 |
| C- I+ | 1.76 | 2.60 | 3.45 | 3.91 | 2.50 | 14.3 | 18.3 | 21.9 | 17.0 |
| C- I+ MQT+ | 2.94 | 3.39 | 4.52 | 3.68 | 4.12 | 7.70 | 8.60 | 9.70 | 10.8 |
| C+ DA- | -0.49 | 3.84 | 12.64 | 12.26 | 11.5 | 36.5 | 97.6 | 99.2 | 98.2 |
| C+ I+ | 0.48 | 6.53 | 15.58 | 14.8 | 14.8 | 37.5 | 81.8 | 81.8 | 97.8 |
| C+ I+ MQT+ | 1.35 | 25.65 | 48.70 | 46.78 | 44.9 | 52.6 | 98.7 | 97.4 | 97.7 |
| | | **Total I/O wait ($\%I/O\,wait$)** | | | | **Application performance ($\#\,useful\,work/kWh$)** | | | |
| | | **1** | **2** | **4** | **8** | **1** | **2** | **4** | **8** |
| C- DA- | | 45.4 | 27.6 | 10.2 | 53.5 | 6559 | 35832 | 43687 | 38519 |
| C- I+ | | 42.7 | 49.3 | 51.3 | 64.3 | 15755 | 20264 | 22985 | 15124 |
| C- I+ MQT+ | | 45.3 | 57.7 | 58.8 | 63.2 | 21184 | 29207 | 23726 | 26587 |
| C+ DA- | | 32.5 | 1.0 | 0.5 | 0.97 | 19230 | 41138 | 41573 | 39054 |
| C+ I+ | | 19.4 | 11.1 | 5.2 | 1.3 | 31859 | 57713 | 54830 | 53007 |
| C+ I+ MQT+ | | 0.5 | 0.5 | 1.5 | 2.4 | 100568 | 162348 | 158575 | 154786 |
| | | **Application energy efficiency ($W/\#\,of\,transaction$)** | | | | **System energy usage ($kWh$)** | | | |
| | | **1** | **2** | **4** | **8** | **1** | **2** | **4** | **8** |
| C- DA- | | 0.15 | 0.03 | 0.02 | 0.02 | 0.17 | 0.23 | 0.26 | 0.19 |
| C- I+ | | 0.06 | 0.05 | 0.04 | 0.07 | 0.17 | 0.17 | 0.17 | 0.17 |
| C- I+ MQT+ | | 0.05 | 0.03 | 0.04 | 0.04 | 0.16 | 0.16 | 0.16 | 0.15 |
| C+ DA- | | 0.05 | 0.02 | 0.03 | 0.03 | 0.2 | 0.3 | 0.29 | 0.29 |
| C+ I+ | | 0.03 | 0.02 | 0.02 | 0.02 | 0.21 | 0.27 | 0.27 | 0.28 |
| C+ I+ MQT+ | | 0.01 | 0.01 | 0.01 | 0.01 | 0.26 | 0.3 | 0.29 | 0.29 |

Table 3: Summary of results in each scenario

## 4 Results and Discussion

As a reference point, idle system measurements are as follows: Total CPU usage rate is 0.09 %, total I/O wait is 0.09 %, and system power usage is 28W. All scenario measurements are shown in Table 3. We discuss our findings below.

## 4.1 IT Resource Usage Results

### 4.1.1 CPU Usage Rate

The percentage of CPU usage indicates how much of the processors capacity is currently in use by the system and is, therefore, a key contributor to the systems total energy usage. In the case of a multicore processor, utilization is computed as the average usage of all cores. Utilization is split into the following categories: 1) "%user" shows the percentage of CPU utilization that occurred while executing at the user level (application), 2) "%nice" shows the percentage of CPU utilization that occurred while executing at the user level with "nice" priority, 3) "%system" shows the percentage of CPU utilization that occurred while executing system-level (kernel) tasks, and 4) "%I/O wait" shows the percentage of time that the CPU was idling while waiting for completion of disk I/O requests [7].

Irrespective of number of users, the compression feature (C) dramatically (3 to 5 times) increased CPU usage (Figure 2a). Considering Scenario 1 with one connected user, for example, CPU usage was 10.5%, while in Scenario 4, it was 36.5%. Indexes suggested by the Design

[7]IOSTAT manual. http://sebastien.godard.pagesperso-orange.fr/man_iostat.html

Advisor (I) had only negligible effect on CPU usage with one user (Scenario 1 vs. Scenario 2). However, for multiple users, reduction of CPU usage was correlated with DA using Indexes (Scenario 1 vs. Scenario 2) and *I+MQT* (Scenario 1 vs. Scenario 3). It is noteworthy that the inclusion of all features into the system also increased CPU usage rate (Scenario 1 vs. Scenario 6).

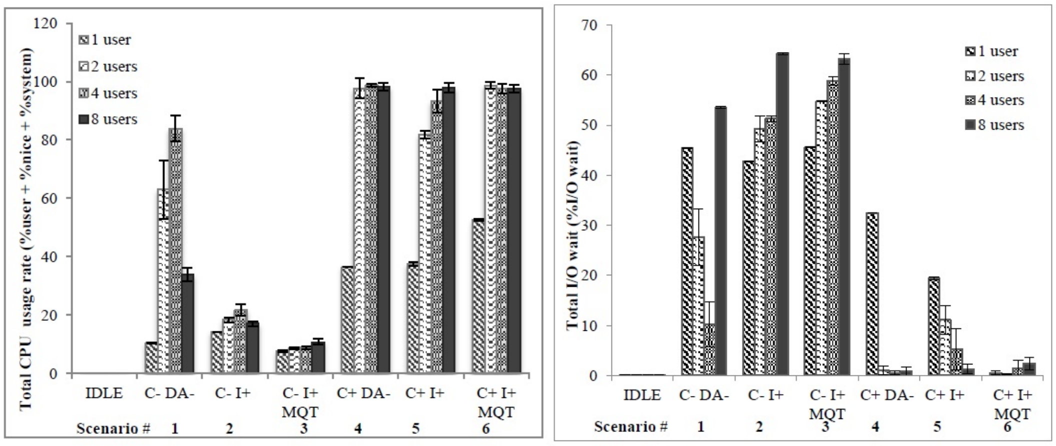


Figure 2: (a) Total CPU usage rate(%) (%user+%nice+%system) with 1-8 user(s) connection (b) Total I/O waits (%) with 1-8 user(s) connection.

When compression *(C)* was combined with the indexes and MQT feature *(I+MQT)*, the effect on CPU usage yielded peculiar results. Comparing Scenario 3 and Scenario 6, for instance, dramatic differences in CPU usage rate can be observed. On the other hand, by simultaneously combining all three features, the number of statements processed per unit time increased (see Table 3). This is also an indication of the increased performance. When we look at the results of multiple users connected to the database, CPU usage rate decreased as the number of users increased. For instance, the inclusion of compression to the system with one user increased the CPU usage rate by 4 times; however, with two and four users, CPU usage rate was increased by 1.8 and 1.1 times, respectively. It will be recalled that compression is designed to increase CPU utilization to benefit smaller databases. This explains why CPU utilization increased more with only one user. On the other hand, with multiple users, the processor is divided among the users. However, even in this case, compression still helps increase CPU utilization, as the processor services one user at a time before moving on to the next user.

In Scenario 4, CPU usage rate reaches a load of almost 100% as the number of concurrent users reaches two. However, Scenarios 1, 2, and 3 do not utilize the CPU as much. This can be explained by the I/O bottleneck, as shown by I/O-wait data in Table 3. In fact, the increasing number of users in Scenario 1, 2 and 3 leads to a reduction of CPU load because the CPU has to wait longer for data from the hard drive as a result of increased concurrency. On the other hand, in Scenario 5 and 6, the CPU usage rates are high again because the processor is managing more simultaneously active features.

In the case of eight users, the CPU usage rate has increased again by approximately 4 times when comparing Scenario 1 to Scenarios 4, 5 and 6. The intuitive justification for this result is that the complexity of queries is high, even though the number of users is not increased to a large number. Idle time may be ignored based on its small value (Table 3). In addition, adaptive compression diminishes CPU usage in idle time since it compresses and transmits blocks on-the-fly.

#### 4.1.2 I/O Wait

The data shown in Figure 2b reveal that all scenarios without the compression feature (Scenarios 1, 2, and 3) are I/O-bound. Total I/O-wait statistics for Scenarios 4, 5 and 6, in which the compression feature is enabled, shows significant reduction of I/O wait time compared to the remaining scenarios with compression disabled. Without the compression feature, we did not observe any decrease in I/O-wait, even with the other two features enabled (Scenarios 1, 2 and 3). When we compare statistics for Scenarios 4, 5 and 6, as shown in Figure 2b, we see that compression yields better results for multiple users. This can be explained by the fact that the system leverages compressed database data, which is cached in memory, to process concurrent requests from the users. Without any object ($I$ or $MQT$) feature, the system performed faster (Scenarios 4 vs. 5 and 6). In Scenario 5, the compression feature was enabled, but I/O time was still slightly increased by enabling the Index feature. With one user, Scenario 4 shows slight overhead, and this put more pressure on the system. Enabling indexes *(I)* and *(I+ $MQT$)* helped to reduce the time (Scenario 5 and 6) and helped to fit all the data into memory. Results reveal that the compression feature alone reduced the I/O wait time significantly. We also observed a significant decrease in I/O wait time when all the features were enabled (Scenario 2 vs. scenario 6), which is a good indication of feature interaction.

Large I/O wait (%) indicates that all the data stored in the database could not fit into the system's memory. Data which do not fit into memory cause high volumes of read and write requests issued to the hard drive. This is the case for the first three scenarios. Low I/O wait time in Scenarios 4, 5 and 6 may be explained by a significant reduction in the amount of data that must be loaded into memory. By combining data compression, the use of indexes instead of raw data for some operations and precomputing partial record sets via $MQT$, the database engine was able to load most of the required data into memory. This led to a decrease in I/O wait by an order of magnitude. It is expected that the addition of the objects suggested by DA will improve database performance. Although CPU usage rate in Scenario 6 is the highest (Figure 2a), it should not be ignored that the average statement count per unit of time in Scenario 6 is also the highest (Table 3) and that the total I/O wait is the lowest (Table 3). More users mean more requests which implies more workload for the CPU, less idle time, and low I/O-wait.

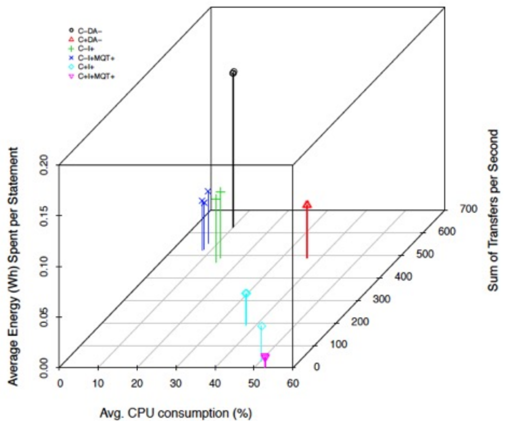
#### 4.1.3 Energy consumption vs. CPU and I/O utilization

In this subsection we will analyze the relation between system statistics and energy consumption for all the experiments. This will help us understand general constraints of the system under study.

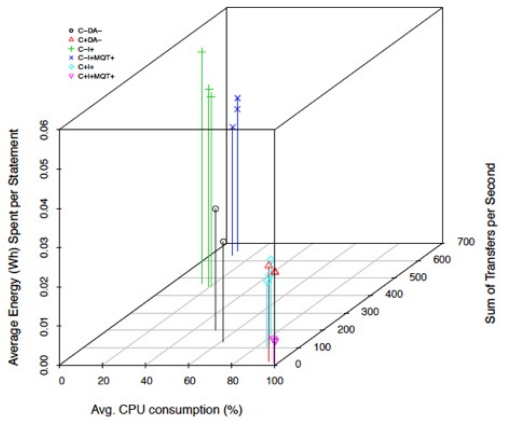
Figure 3 shows the relation between energy consumption, CPU utilization and I/O load. We compute CPU utilization and I/O load as follows.

As shown in Figure 3, the more I/O operations one has to do, the more CPU has to idle waiting for the I/O operations to complete. This leads to increased time spent and energy consumption, as the CPU idles, waiting for the data to be read from (and written to) the hard drive.

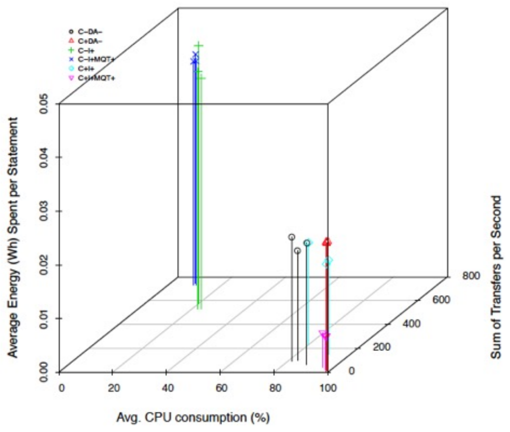
As seen from Figure 3, if the database can load all the data into memory, then it requires minimal amount of I/O operations, because the whole database is cached into memory – no access to the hard drive is required. In this case the workload often becomes CPU-bound, as the architecture of the database engine (for the releases under study) cannot effectively parallelise



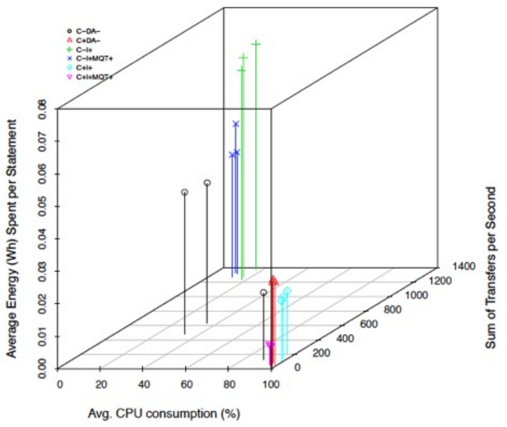
(a) Connection count = 1



(b) Connection count = 2



(c) Connection count = 4



(d) Connection count = 8

Figure 3: Relation between CPU utilization, hard drive utilization, and energy consumed for different number of connections. Each data point represents a single workload execution. Different point types denote different feature sets, as shown on the legend and summarized in Table 2. Vertical lines from the points are projections from data points to horizontal plane.

processing of a single query. An example of this case is shown in Figure 3a, where we simulate a single user connected to the database engine: Scenario 6 (C+I+MQT) does not require access to the hard drive and fully utilizes one core of the CPU.

If the database engine cannot load all the data into memory and has to wait for I/O operations to complete, then the workload becomes I/O bound. The number of I/O operations increases, while CPU utilization decreases. An example of this case can be seen in Figure 3(a) for Scenario 1, where CPU utilization drops to $\approx 10\%$, while Transfers per Second increases to $\approx 600$.

To better understand this phenomenon, we plot statements per second vs. energy consumed in Figure 4a. We also plot statements per second vs. average power consumed per statement Figure 4b. The average power consumed is computed by dividing the amount of energy consumed by the time spent. From hereon, for the sake of brevity, we will use the term 'power' instead of ‘average power’.

Figure 4b shows that the more statements we process per unit of time, the more power we consume. However, the more statements we process – the less energy we consume. This can be explained by differences in the amount of power consumed by CPU and HDD.

Both CPU and HDD require different amount of power while idling or under load. However, the amount of power (in absolute numbers) is vastly different: our HDD consume 0.4-0.5W while idling and 1.2-1.5W under load [8][9]; our CPU, on the other hand, has thermal design power of 35W. Note that thermal design power is not equivalent to power consumed by the CPU. It gives the amount of power that CPU dissipates while being fully active. However, this value give us an understanding of the magnitude of power consumption.

The higher the CPU utilization is – the higher its power consumption. From Figure 3 we know that experiments which consumed the least amount of energy (and thus were able to process more statements per second) had high CPU utilization. The figure also shows that these workloads had almost no I/O activity, since all the data was loaded into memory. CPU did not have to idle waiting for I/O operations to complete and was able to maintain high utilization rate. These experiments correspond to data points in the top-right corner of Figure 4(b),where power consumption was in the range of 60W.

As mentioned above, HDD consumes significantly smaller amount of energy than CPU (compare 1.5W with 35W). This becomes important for experiments causing high I/O utilization. Hard drive power consumption will remain at ≈1.5W, while CPU utilization will drop significantly (as shown in Figure 3), as CPU idles waiting for I/O operations to complete. The cases of experiments with idling CPU are represented by data points in bottom-left corner of Figure 4, where power consumption was ≈30W, which is quite close to the baseline power consumption of 28W. Even though power consumption in these cases is low, the energy consumption is high, since we have to integrate low power consumption value over prolonged time interval (when the CPU waits for I/O operations to complete). To summarize, the workload configurations that had the highest performance were the CPU-bound ones (consuming the least amount of energy and the largest amount of power), while the workload configurations with the lowest performance were the I/O-bound ones (consuming the largest amount of energy and the least amount of power).

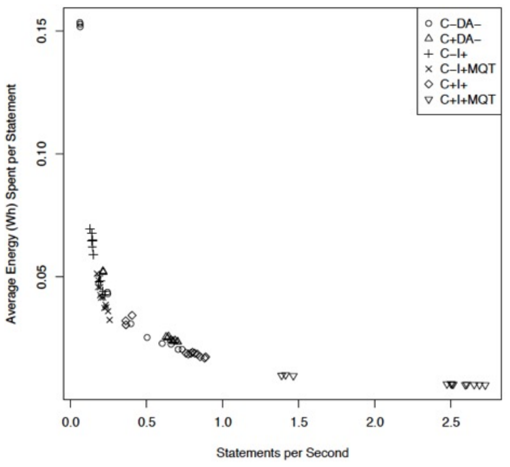
Will these trends hold on other computers? From hard drive to hard drive and from CPU to CPU the power consumption will vary. However, as long as the general principle of a hard drive consuming significantly less power than CPU remains – the trends would hold and would be transferable to other computers. For example, consider the latest (at the time of writing) server hardware: Western Digital Re 1TB WD1003FBYZ HDD, designed for data centers and Intel Xeon E3 1280v5 CPU. The hard drive consumes 5.9W while idling and 8.6 W under load [10]. The CPU thermal design power is 80W [11]. Both hard drive and CPU consume more power than the ones in our setup (compare 1.5W with 5.9 W and 35W with 80W). However, the general principle remains – while fully utilized, hard drive will need significantly less power than CPU.

[8]http://toshiba.semicon-storage.com/product/storage/pdf/hdd18_29.pdf

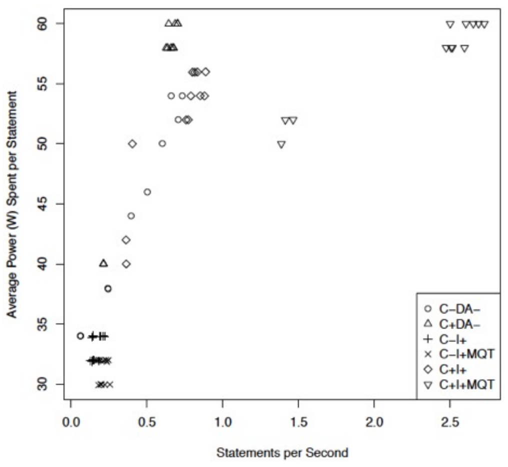
[9]http://www.tomshardware.com/reviews/1.8-toshiba-hdd,2576-10.html

[10]WD Re Datacenter Capacity HDD http://www.wdc.com/wdproducts/library/SpecSheet/ENG/2879-800044.pdf

[11]Specifications: http://ark.intel.com/products/88171/Intel-Xeon-Processor-E3-1280-v5-8M-Cache-3_70-GHz



(a) Statements per Second vs. Average Energy



(b) Statements per Second vs. Average Power

Figure 4: Relation between statements per second and energy (average power) consumed for all experiments. A data point represents the time and average energy (average power) spent per statement data gathered for a given run of an experiment.

### 4.1.4 Storage Usage

We chose the amount of storage space consumed in Scenario 1, in which none of the software features is enabled, as a reference point. All but one scenario consumed more space than the reference (see Figure 5a). The only exception is Scenario 4 with only the compression feature enabled. The remaining scenarios needed extra space to create additional objects, i.e., indices and MQTs. The results also showed a trade-off between CPU and storage usage. In the case of Scenario 3, total CPU usage rate was the lowest (Figure 2a), while storage usage was the highest. Because of the compression feature, useful unit of work increased. This tended to increase average statement count per time (Table 4) and total CPU usage rate (Figure 2a). Thus, the system requires more storage, suggesting the necessity of considering a trade-off between CPU consumption and storage. A good example of such trade-off is given in Scenario 4 (Figures 2a and 5a).

## 4.2 Lifecycle Metric-Application Performance (AP)

*AP* is defined as the amount of work per unit of energy consumed. The increase of *AP* in the last three scenarios, especially Scenario 6, arises from the increase of average statement per time and average CPU usage rate. The increased rate of average statement grows faster than the CPU usage rate. Theoretically, if the workload size is fixed, then hardware is dedicated only to this workload, and the efficiency of the CPU usage rate is generally ignored. Although enabling all the features may be seen as not beneficial (Figure 2a, Scenario 6), we observe that this is not the case for Scenario 6, as shown in Figure 5b. *AP* reaches its maximum value when all the features are enabled, suggesting the benefit of enabling all features.

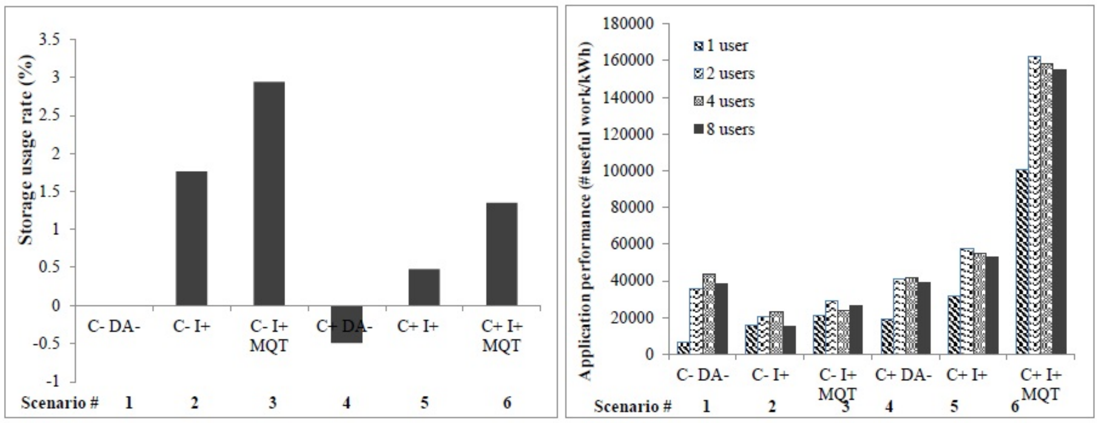


Figure 5: (a) Storage usage rate (%) (b) Application performance (number of transaction/kWh) results with 1-8 user(s) connection.

## 4.3 Energy Impact Metrics

### 4.3.1 Application Energy Efficiency (AEE)

*AEE* characterizes the required energy for processing a single statement [10]. The total energy consumed is the sum of the energy to complete the entire workload. The increase in the average statement count per unit of time leads to the decrease in the energy usage per statement (Table 3). Figure 6a illustrates the energy efficiency (kWh/amount of useful work) of the application in each scenario. We observed results similar to those of *AP* in that the cumulative effect of all three features is significant when Scenario 1 is compared to Scenario 6, especially with one user. When the number of users increases, the amount of useful unit of work also increases. The significant benefit of this is apparent in Scenario 1, in which all features are disabled. The most energy-efficient scenario is Scenario 6, in which we can see the cumulative effect of all interacting features. In Scenarios 5 and 6, we observe that *AEE* stays the same when users are greater than, or equal to, 2. This happens because CPU resources are exhausted.

### 4.3.2 System Energy Usage

Figure 6b illustrates the total energy usage, as measured in kWh, for the software system in each scenario. An energy savings can be clearly seen when enabling both indexes *(I)* and *MQT* features together (Scenario 1 vs. scenario 3). On the other hand, enabling the compression feature alone seems to boost system energy consumption. As discussed in Section 4.1.2, the addition of the objects suggested by *DA* improves database performance. It also helps to reduce system energy consumption (Scenario 4 vs. Scenarios 5 and 6).

As noted above, this work aimed to identify software feature interactions and the cumulative effects of these interactions on energy consumption of the system. Accordingly, Table 4 was refined from the overall results of Table 3 to show cumulative effect with all features enabled. Irrespective of number of users, energy efficiency improved when all features are enabled. However, it can also be seen that cumulative effect increases the systems energy usage. *AEE* depends on the number of statements, and because the compression feature is enabled, the number of statements increases. Therefore, *AEE* increases. On the other hand, EC increases

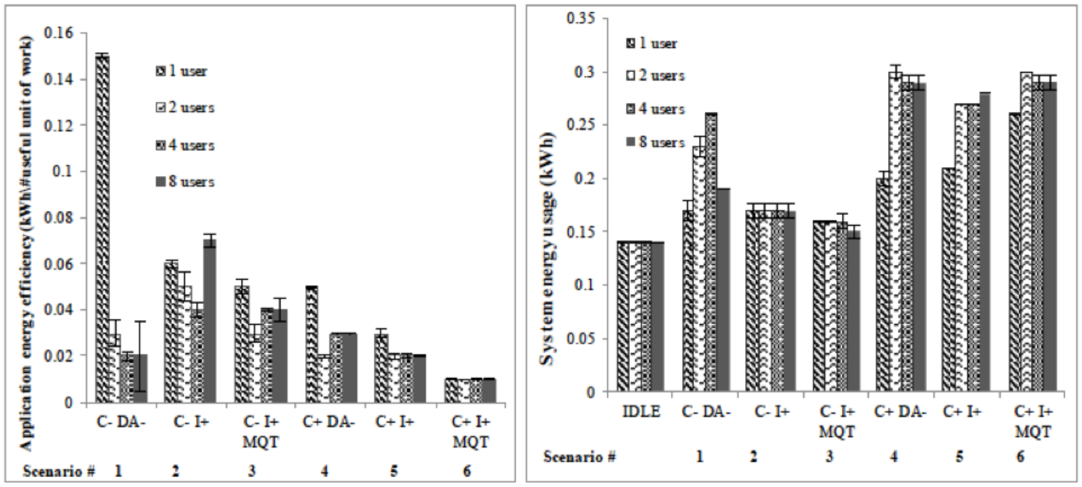


Figure 6: (a) Application energy efficiency (kWh/amount of useful work) results with 1-8 user(s) connection. (b) System energy usage results with 1-8 user(s) connection.

with increasing units of work (Section 3.2).

| | C- DA+ | C- I+ MQT+ | C+ I+ MQT |
|---|---|---|---|
| **CPU** | 10.1 | 7.7 | 52.6 |
| **I/O wait** | 45.4 | 45.3 | 0.5 |
| **Application energy efficiency** | 0.15 | 0.05 | 0.01 |
| **System energy usage** | 0.17 | 0.16 | 0.26 |

Table 4: Cumulative effect, one user

### 4.3.3 Research Questions Evaluation

*RQ1: Evaluation of Energy Efficiency*

The chosen green metric set is simple and easy to apply. This set identifies various parts of the software system where energy is consumed. Therefore, it will be useful in evaluating the energy consumption of software based on the outcome of the system. However, further refinements may be needed according the operational behaviour of the system during the measurement period. In the future, this metric set would be a candidate for use when giving a green label to a software system.

*RQ2: Cumulative Effect of Feature Interaction*

Experiments showed the potential interaction points, especially those that are undesirable, in different scenarios. For example, from Table 4, it can be seen that the compression feature acts as an enabler for $I$ and $MQT$ such that without compression, $I$ and $MQT$ actually slow down the system. Table 4 also shows that feature interaction improves the software's energy efficiency. In this respect, the proposed approach may be considered as a preliminary model setting forth a qualitative paradigm that will allow developers to decide which new features to add to existing software systems.

## 4.4 Threats to Validity

*Reliability:* Measurement error is a concern for modern systems, as well as the overhead of the test bed. Therefore, we executed the workload against each configuration four times to estimate measurement error and, hence, validate the reliability of results.

*Internal Validity:* The first limitation is selection bias in choosing the database software system. However, our software, IBM DB2 and its corresponding features, are well known and widely used in business environments. Our second limitation is the measurement of energy consumption. The total energy was consumed not only by the execution of the workload, but also by the operating system. It is difficult to measure energy consumption of a given software product in isolation. In order to address this threat, the database was used only with the essential services, and no additional applications were executed in parallel. We also measured the baseline power consumption of the system. Our last limitation involves the number of users connected to the test database. In a production environment, a database engine may need to withstand a higher number of users. However, the number of users in our study (eight) is proportional to the size of our test computer, which has only two CPU cores. Moreover, the impact of parallel requests not the primary aim of this study.

*Construct Validity:* One distinct threat arises from inconsistency of workloads under study. To address this threat, we used the same standard TPC-H workload to ensure consistent execution in all runs and, hence, more accurate results. Another threat is the choice of metrics. Addressing this threat, we used a set of metrics that are well known in the literature (Section 2.3), and all the required measurements were done. Additionally, we ran each experiment four times to get various readings for the measurements and avoid measurement errors, which helped to ensure the correctness of the data results.

*External Validity:* We are not analyzing the production environment. The testing environment is a laptop, which is tuned to minimize electricity consumption, sacrificing efficacy with consumer-grade operating system. However, we show that as long as the general principle of a hard drive consuming significantly less power than CPU remains – the trends would hold and would be transferable to other computers, including production servers.

It is difficult to draw general conclusions from empirical studies in software engineering. Our results are limited to the analyzed data and the context. We studied one database product. While generalization to other software products is not possible for obvious reasons, the software system represents a critical case [33] of a relational database management system. Our experimental framework could, therefore, be applied to other projects with well-designed and controlled experiments. In this study we do not aim to build a theory, rather we would like to have a deeper understating of the impact of features on energy consumption in a database software product. However, the concepts can be easily applied to other software projects.

# 5 Conclusion

In this paper, we have shown the individual and cumulative effects of software features on the energy efficiency of a system, and metrics were used to measure system performance. The adaptive data compression feature of DB2 utilizes a number of compression techniques, including table-wide and age-wide compression [7], leading to significant reduction of storage space. This feature speeds up I/O-intensive workload. However, its usage leads to CPU overhead associated with compression and decompression of the data. Both $I$ and $MQT$ need and consume extra disk space. Reading data from disk to memory for processing is one of the slowest database operations. Disk storage of compressed data leads to fewer I/O operations needed to retrieve or store the same amount of data in comparison with the uncompressed dataset. Thus, for disk I/O-bound workloads when the system is either idling or waiting for data to be accessed from the disk, query processing time may be noticeably improved. Furthermore, DB2 processes buffer data in memory in its compressed form, thereby reducing the amount of memory consumed compared to uncompressed data. This has the effect of immediate increase in the amount of memory available for the database without increasing physical memory capacity. In turn, this frees up additional memory for other database or system operations. This further improves database performance for queries and other operations.

We have seen that the compression ($C$) feature and the indexes and MQT feature ($I$+$MQT$) dramatically increase CPU usage rate when they are enabled simultaneously, leading to improved engine performance. This may a positive impact on energy efficiency of the system, making a combination of all three features the greenest. Note the existence of a trade-off: these features requires additional storage space.

The features that we selected are not "endemic" to the DB2 product (even though implementation of the features would differ from manufacturer to manufacturer); most popular relational database engines have them implemented. Therefore, our findings would be beneficial to both practitioners and researchers. Software designers and developers may use our methodology and findings to select the most useful combination of features for energy efficiency and performance. They may use them to program features in different combinations, tailoring the design to specific needs, to reuse features in different design variants, to explore potential feature interactions, and to find and fix undesired interactions. Moreover, developers may assess the energy impact and performance of their product with the help of our metric set.

Going forward, we will perform new empirical analysis of transformation and enhancement of features during system evolution. These will reveal additional solid findings for trade-off models related to energy efficiency.

## Acknowledgement

This research is, in part, supported by NSERC Discovery Grant no. 402003- 2012. We would like to thank the IBM Canada Lab, Toronto, for making their development data available for research and providing strategic help during all phases of this research. The opinions expressed in this paper are those of the authors and not necessarily those of IBM Corporation.

## References

[1] Sven Apel, Joanne M Atlee, Luciano Baresi, and Pamela Zave. Feature interactions: The next generation. 2014.

[2] Sven Apel and Christian Kästner. An overview of feature-oriented software development. *Journal of Object Technology*, 8(5):49–84, 2009.

[3] Jeroen Arnoldus, Joris Gresnigt, Kay Grosskop, and Joost Visser. Energy-efficiency indicators for e-services. In *Green and Sustainable Software (GREENS), 2013 2nd International Workshop on*, pages 24–29. IEEE, 2013.

[4] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *IEEE computer*, 40(12):33–37, 2007.

[5] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. The goal question metric approach. In *Encyclopedia of Software Engineering*. Wiley, 1994.

[6] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future generation computer systems*, 28(5):755–768, 2012.

[7] Bishwaranjan Bhattacharjee, Lipyeow Lim, Timothy Malkemus, George Mihaila, Kenneth Ross, Sherman Lau, Cathy McArthur, Zoltan Toth, and Reza Sherkat. Efficient index compression in DB2 LUW. *Proceedings of the VLDB Endowment*, 2(2):1462–1473, 2009.

[8] Eugenio Capra, Giulia Formenti, Chiara Francalanci, and Stefano Gallazzi. The impact of mis software on it energy consumption. In *ECIS*, pages 1–13, 2010.

[9] Andreas Classen, Patrick Heymans, and Pierre-Yves Schobbens. What?s in a feature: A requirements engineering perspective. In *Fundamental Approaches to Software Engineering*, pages 16–30. Springer, 2008.

[10] Transaction Processing Performance Council. TPC benchmark-H, decision support, standard specification, (accessed August, 2016). http://www.tpc.org/tpch/spec/tpch2.14.4.pdf.

[11] Transaction Processing Performance Council. TPC-energy specification, standard specification, version 1.5.0, (accessed August, 2016). http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-energy_v1.5.0.pdf.

[12] Markus Dick and Stefan Naumann. Enhancing software engineering processes towards sustainable software product design. *EnviroInfo*, pages 6–8, 2010.

[13] Mireille Faist Emmenegger, Rolf Frischknecht, Markus Stutz, Michael Guggisberg, Res Witschi, and Tim Otto. Life cycle assessment of the mobile communication system umts: towards eco-efficient systems (12 pp). *The International Journal of Life Cycle Assessment*, 11(4):265–276, 2006.

[14] Lorenz M Hilty and Wolfgang Lohmann. The five most neglected issues in ?green it? *CEPIS Upgrade The European Journal for the Informatics Professional*, 12:11–15, 2011.

[15] Lorenz M Hilty, Wolfgang Lohmann, Siegfried Behrendt, Michaela Evers-Wlk, Klaus Fichter, and Ralph Hintemann. ICT for sustainability: An emerging research field. Technical report, Technical Report (UBA-FB) 001883/2,E, 2015.

[16] Abram Hindle. Green mining: A methodology of relating software change and configuration to power consumption–web edition. *Web Edition*, 2014. http://webdocs.cs.ualberta.ca/~hindle1/2014/green-emse-web-edition.pd.

[17] Georgios Kalaitzoglou, Magiel Bruntink, and Joost Visser. A practical model for evaluating the energy efficiency of software applications. In *ICT for Sustainability 2014 (ICT4S-14)*. Atlantis Press, 2014.

[18] Kyo C Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh. Form: A feature-; oriented reuse method with domain-; specific reference architectures. *Annals of Software Engineering*, 5(1):143–168, 1998.

[19] Eva Kern, Markus Dick, Timo Johann, and Stefan Naumann. Green software and green it: An end users perspective. In *Information Technologies in Environmental Engineering*, pages 199–211. Springer, 2011.

[20] Alexander Kipp, Tao Jiang, and Mariagrazia Fugini. Green metrics for energy-aware it systems. In *Complex, Intelligent and Software Intensive Systems (CISIS), 2011 International Conference on*, pages 241–248. IEEE, 2011.

[21] Sedef Akınlı Koçak, Andriy Miranskyy, Gülfem Işıklar Alptekin, Ayşe Başar Bener, and Enzo Cialini. The impact of improving software functionality on environmental sustainability. *on Information and Communication Technologies*, page 95, 2013.

[22] Rachita Kothiyal, Vasily Tarasov, Priya Sehgal, and Erez Zadok. Energy and performance evaluation of lossless file data compression on server systems. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, page 4. ACM, 2009.

[23] Jia Liu, Don Batory, and Christian Lengauer. Feature oriented refactoring of legacy applications. In *Proceedings of the 28th international conference on Software engineering*, pages 112–121. ACM,

2006.

[24] Sara S Mahmoud, Imtiaz Ahmad, et al. Green performance indicators for energy aware it systems: Survey and assessment. *Journal of Green Engineering*, 3(1):33–69, 2012.

[25] A Miransky, Sedef Akinli Koçak, Enzo Cialini, and Ayse Basar Bener. Save energy with the DB2 10.1 for linux, unix, and windows data compression feature. *IBM DeveloperWoks, Technical Library*, 2013.

[26] Andriy Miranskyy, Zainab Al-zanbouri, David Godwin, and Ayşe Başar Bener. Database engines: Evolution of greenness, 2017. Journal of Software Evolution and Process, Accepted.

[27] San Murugesan. Harnessing green it: Principles and practices. *IT professional*, 10(1):24–33, 2008.

[28] Stefan Naumann, Markus Dick, Eva Kern, and Timo Johann. The greensoft model: A reference model for green and sustainable software and its engineering. *Sustainable Computing: Informatics and Systems*, 1(4):294–304, 2011.

[29] Giuseppe Procaccianti. Energy-efficient software. *Amsterdam: Vrije Universiteit*, 2015.

[30] David Salomon. *Data compression: the complete reference*. Springer Science & Business Media, 2004.

[31] Wolfram Scharnhorst, Lorenz M Hilty, and Olivier Jolliet. Life cycle assessment of second generation (2g) and third generation (3g) mobile phone networks. *Environment international*, 32(5):656–675, 2006.

[32] Vivek Tiwari, Sharad Malik, and Andrew Wolfe. Power analysis of embedded software: a first step towards software power minimization. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 2(4):437–445, 1994.

[33] Robert K Yin. *Case study research: Design and methods*. Sage publications, 2013.

[34] Daniel C Zilio, Jun Rao, Sam Lightstone, Guy Lohman, Adam Storm, Christian Garcia-Arellano, and Scott Fadden. DB2 design advisor: integrated automatic physical database design. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 1087–1097. VLDB Endowment, 2004.