An Evaluation of the Impact of End-to-End Query Optimization Strategies on Energy Consumption

Eros Cedeño[1], Ana Aguilera[2], Denisse Muñante[3], Jorge Correia[1], Leonel Guerrero[1], Carlos Sivira[1] and Yudith Cardinale[1,4]

[1]*Departamento Computación y Tecnología de la Información, Universidad Simón Bolívar, Caracas, Venezuela*
[2]*Escuela de Ingeniería Informática, Facultad de Ingeniería, Universidad de Valparaíso, Chile*
[3]*ENSIIE & SAMOVAR, Évry, France*
[4]*Centro de Estudios en Ciancia de Datos e IA (ESenCIA), Universidad Internacional de Valencia, Spain*

Keywords: Energy Consumption, Relational Databases, Empirical Evaluation, Query Optimization.

Abstract: Query optimization strategies is an important aspect in database systems that have been mainly studied only from the perspective of reducing the execution time, neglecting the analysis of their impact on energy consumption. We perform an empirical evaluation for understanding the impact of end-to-end query optimization strategies on the power consumption of database systems, from both client and server perspectives. We perform tests over a PostgreSQL database for two optimization strategies (*i.e.,* indexation and data compression) using the TPC-H benchmark, configured with 22 queries on a 1GB dataset. To measure the energy consumption of both client and server, we propose Juliet, a C++ agent for monitoring and estimating Linux processes energy consumption in Joules (J). Experimental results show that *indexation* is more effective than *data compression* to reduce the energy consumed by the execution of the majority of the 22 queries tested.

1 INTRODUCTION

Database Management Systems (DBMS), specifically Relational Database Management Systems (RDBMS), are software components that could require a lot of hardware resources for a database with a lot of information, and this possibly could lead to a lot of power consumption at several modules on the servers, as well as on the client side.

Currently, there exist studies approaching the energy-efficiency of RDBMS based on prediction models (Guo et al., 2022), by proposing guidelines to reduce energy waste in terms of improving query planning, processing, and execution (Stavros, 2009; You et al., 2020), or by measuring the energy consumption with tools and benchmarks (Guo et al., 2022). In particular, query optimization strategies is an important aspect in database systems that have been studied only from the perspective of performance improvements in their operations, rather than on the energy costs of their operations. Nevertheless, the goal of the query optimization would move to choosing the best trade-offs between different metrics in order to meet the multi-criteria quality of service that consumers have requested. It is important to look into the potential trade-offs between various metrics and resources, as well as the intricate relationships between these metrics and how they affect query processing as a whole (Guo et al., 2022).

To better understand the impact of end-to-end query optimization strategies on the power consumption of RDBMS, from both client and server perspectives, we perform an empirical evaluation by considering two optimization strategies (*i.e.,* indexation and data compression) and their combination. We use TPC-H benchmark[1], configured with 22 queries on a 1GB dataset, to do tests over a PostgreSQL RDBMS. To monitor the power consumption of both client and server, we propose Juliet, a C++ agent aimed to monitor and estimate the energy consumption of Linux processes. Experimental results show that *indexation* is more effective than *data compression* to reduce the energy consumed by the execution of the majority of the 22 queries of the TPC-H benchmarking. The com-

[1]https://www.tpc.org/tpch/default5.asp

Cedeño, E., Aguilera, A., Muñante, D., Correia, J., Guerrero, L., Sivira, C. and Cardinale, Y.
An Evaluation of the Impact of End-to-End Query Optimization Strategies on Energy Consumption.
DOI: 10.5220/0012708000003687

In *Proceedings of the 19th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2024)*, pages 657-665
ISBN: 978-989-758-696-5; ISSN: 2184-4895

Table 1: Related work. *Where: LSSVM = Least Square Support Vector Machine, PM = Predictive Model.*

Work	Measure	Method	DBMS	Benchmark	Optimization strategy/Operations	Side
(Yang et al., 2023)	Energy	LSSVM, sensors, PM	Oracle 11g	TPC-H	-	Client (C)
(Karyakin and Salem, 2017)	Background and active power	SIMM riser and linear power model	Shore-MT, MonetDB	TPC-C, TPC-H	-	Server (S)
(Mahajan et al., 2019)	Performance, power, energy	Self-built power measurement tool	MySQL, MongoDB, Cassandra	YCSB Twitter data	Index, ordered writes, aggregations, row caching, compaction	Client
(Koçak et al., 2018)	Average power	UPM EM100 Energy Meter	IBM DB2	TPC-H	Compression, materialized query tables, index	Client
(Lella et al., 2023)	Energy, Mean Energy	PSUTIL, python package	MySQL, PostgreSQL, MongoDB, Couchbase	Netflix Userbase, SMS Spam coll.	SELECT, INSERT, DELETE, UPDATE	Client
(Xu et al., 2015)	Energy	RLS, Wattsup power, and PM	PostgreSQL	TPC-H, SDSS	-	Server
(Rodríguez et al., 2013)	Energy, power, time	Wattsup power	PostgreSQL	TPC-DS	-	Server
(Roukh et al., 2016)	Time, power (W)	Watts Up Pro	PostgreSQL	TPC-H, SSB	-	Server
(Guo et al., 2017)	Time, energy (J), power (W)	WT3000, power analyzer, and PM	Oracle 11g	TPC-C, TPC-H	Buffer Cache, Shared Pool	Server
(Liu et al., 2013)	Time, power	PM1000, power analyzer, and PM	PostgreSQL 9.1.8	TPC-H	-	Server
(Procaccianti et al., 2016)	Time, energy	Watts Up Pro	PLAIN SQL, Propel, and TinyQueries	-	CREATE, READ, UPDATE, DELETE	Client
Our proposal	**Time, energy**	**Juliet**	**Postgres**	**TPC-H**	**Index, compression**	**C & S**

bination of indexation and data compression does not outperforms the *indexation*.

2 RELATED WORK

To better understand the different components of DBMS that impact on the energy consumption, it is necessary to consider several aspects of the database system in the evaluation process. In particular, we consider that the hardware components (*e.g.,* CPU, memory, disk), the optimization strategies, and the evaluated execution side (client or server) are key aspects that might be taken into account during the evaluation process. In this section, we describe recent studies focused on at least one of these aspects, whether based on prediction models for energy consumption estimations (Yang et al., 2023; Karyakin and Salem, 2017), based on empirical evaluation of optimization strategies and operations using measure tools or benchmarks (Mahajan et al., 2019; Koçak et al., 2018; Lella et al., 2023), or based on energy-efficient query optimizers (Xu et al., 2015; Rodríguez et al., 2013; Roukh et al., 2016; Guo et al., 2017; Liu et al., 2013; Procaccianti et al., 2016). Additionally, we look at the measures considered, the method used for evaluation, the DBMS evaluated, and the benchmark used on each study reviewed.

Table 1 summarizes the works reviewed in this section. Estimation models have been successfully used to predict energy consumption with measures such as power and energy workloads and memory usage. Most of the reviewed works are in the context of query optimizers, however, despite their critical significance in decision support systems, there is no evidence that power predictions are used as the primary goal when optimising large join queries (Rodríguez et al., 2013). Our work is more related to those using query optimization strategies, such as indexes or compression. But, these works do not perform an empirical evaluation of the end-to-end server-client application to evaluate the impact of energy consumption of database query optimizations. Furthermore, many of them concentrate the energy consumption analysis on the server side and neglect the analysis on the client side; while the studies that perform the analysis from the client perspective do not carry out a fine analysis to distinguish the measurements by module.

3 STUDY DESIGN

From the the literature review, we realize that only few studies perform empirical assessment to measure the impact of energy consumption in applying different database query optimization strategies (Mahajan et al., 2019; Koçak et al., 2018); however, they do not consider the analysis of the end-to-end execution of

client-server applications. We are interested in evaluating that aspect. Thus, the goal of this study is to *increase* developers awareness about energy consumption for relational databases, from which they can deduce database optimization strategies guidelines *in the context of* the development of client-server applications. From this goal, the following research questions are derived:

RQ1. *Does the implementation of database optimization strategies affect the energy consumption of client-server applications?*

RQ2. *Does the addition of database optimization strategies affect the energy consumption of client-server applications?*

To answer these RQ, we implemented two relational database optimization strategies (*i.e.,* indexation and data compression) on TPC-H and developed client-server Java applications able to intensively access data coming from the TPC-H benchmark. Figure 1 depicts the architectural design of the experiments. For the server side, we use PostgreSQL 16 database system that is open source and is a commonly used for developers and supports indexation and compression as optimization strategies. The TPC-H benchmark, implemented in the server, consists of a suite of business oriented ad-hoc queries and concurrent data modifications. The queries and the data populating the database have been chosen to have broad industry-wide relevance[2]. This benchmark contains 22 queries in total with a dataset with different sizes, from which we select the 1GB dataset. For the implementation of the benchmark in PostgreSQL, we use an implementation that offers a Python script to create the query files and the databases and loads the information of the databases based on the TPC-H benchmark[3]. The Java client application is implemented and configured to execute each of the 22 queries of the TPC-H benchmark.

Our experiments are setup using four different configurations at the server: i) without strategies (called *base*); ii) with compression and without indexation (called *compression*); iii) with indexation and without compression (called *index*); and iv) both compression and indexation enabled (called *index-compression*).

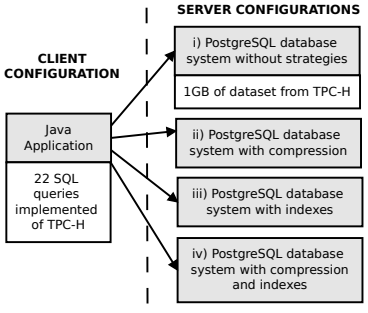

Figure 1: Architectural design of the experiments.

We create the indexes manually following the method provided by Martins et al., who indicate *"create indexes for the columns designated as foreign keys and to the columns in join operations or the columns used under the conditions of WHERE clauses"* (Martins et al., 2021). Regarding the compression, we use the functionality `pg_squeeze` of the PostgreSQL extension[4] to compress databases once they are loaded.

Table 2 shows the storage usage for each configuration. As expected, the configurations with *compression* takes up less storage than other configurations and *indexes without compression* takes the highest storage.

We identify as a hypothesis that the database optimization strategies influence the energy consumed by client-server applications. In order to evaluate the extent of this influence, we collect the energy consumption in Joules (J) through Juliet (see Section 4), at the client and the server. The PostgreSQL application is analyzed during the execution of the query for each query in each iteration and its estimated energy consumption is returned in Joules.

On each test, server and client are executed in the same dedicated computer. The server is an instance of the PostgreSQL service and the client is the Java program used to run all the queries, which is the unique user connected to the server. The experiments were conducted in a 8th generation Intel Core i7 processor, 12 cores, and 4 threads per core, a frequency of 3.75GHz, 16GB of RAM and 500GB of HDD, and Linux Mint 21.1 GNU/Linux Operating System, Java 11, and PostgreSQL 16.

Algorithm 1 presents the scenario of experiments execution, which requires as inputs: i) the list of optimization strategies (*base*, *index*, *compression*, and *index-compression*); ii) the number of iteration for managing the fluctuations values of metrics – *i.e.,* 30 in our experiments; and iii) the list of queries to be executed (*i.e.,* the 22 queries of the TPC-H benchmark).

Per each optimization strategy in the list of strategies (`lstOpt`), we first delete the database if it is already created and we then create the database in the

Table 2: Storage usage for each optimization.

	No compression	Compression
No index	1741 MB	1476 MB
Index	2512 MB	2246 MB

[2]https://www.tpc.org/tpch/default5.asp

[3]https://github.com/Data-Science-Platform/tpch-pgsql.

[4]https://github.com/cybertec-postgresql/pg_squeeze

Require: *lstOpt*; // *base*, *index*, *compression*, *index-compression*
numIterations = 30; // number of iterations
lstQueries; // list of queries (22 queries en total)
Ensure: collect metrics for the analysis of energy consumption of a client-server application
1: **for** *optimization* in *lstOpt* **do**
2: delete the database if it already exists
3: create the database according to the configuration of *optimization*
4: **for** *query* in *lstQueries* **do**
5: **for** *i* in *numIterations* **do**
6: restart the service of database server
7: start monitoring tools
8: execute *query* in the database server with the config.
9: collect metrics (*execution time* from client, *energy consumption* from client and server)
10: stop monitoring tools
11: **end for**
12: **end for**
13: **end for**

Algorithm 1: Scenario of execution of the experiments.

PostgreSQL server taking into consideration the configuration provided by the chosen optimization strategy (line 1 and line 2, in Algorithm 1). For instance, if the strategy is *base*, we create the database without any strategy, whilst if the strategy is *index* we create the database and we then add the indexes, similar to the strategies *compression* and *index-compression*. Once the database is created (line 3), per each query in the list of queries (`lstQueries`), we run 30 executions (line 4 and line 5) in order to manage the energy fluctuations that the energy monitoring tools experience. Per each iteration, we restart the service of the database server in order to have the same conditions for each execution (line 6), we then start the monitoring tools (line 7) and execute the chosen query in the database that was configured with the optimization strategy (line 8). Finally, per each iteration, we collect metrics related to execution time from the client side and energy consumption from the both client and server sides (line 9). This process is repeated for all iterations, queries, and optimization strategies.

4 JULIET: MONITOR OF ENERGY CONSUMPTION

Juliet is a software tool for monitoring the energy consumption of processes executing in a Linux-based operating system. It takes advantage of the RAPL interface provided by Intel processors, which reports an estimate in microjoules of the accumulated energy consumption in real time of the processor in various power domains (*e.g.,* CPU or RAM). Juliet accesses process and CPU usage statistics located in `/proc/` and performs periodic readings at regular and configurable intervals to monitor the percentage of total CPU usage of the system and the percentage of CPU usage allocated to a target process. Based on these percentages and the total energy consumption reported by the RAPL interface, the percentage of the energy corresponding to the target processes is estimated. Juliet is written in C++ Version 11, comprised of four main classes: `RaplLinux` class that manages the interaction with the Linux RAPL interface; `CPU` class for access and compute the CPU statistics, as well as associated functions; `System` class that manages the processes, the system load among other functionalities; and `Monitor` class that handles the monitoring and recording of results. Juliet also searches for all processes that match a given name and analyzes the energy consumption of all of them.

5 RESULTS

To answer the research questions (RQ), we first analyzed the time of execution of the queries for the different configurations with the purpose of assuring that the database optimization strategies have been well-applied on the TPC-H benchmarking. For all of the queries and strategies the expected reduction of the execution time was obtained, however for the space reasons, we do not present these results in this paper. We next perform a statistical analysis to know in which extent the database optimization strategies (negatively or positively) impact the energy consumed by the execution of the 22 queries of the TPC-H benchmarking in the client-server application. Results of this analysis are introduced in Table 3 and Table 4. To get these values, we are guided by a set of steps described as follows:

Step 1: we first analyze the distribution of the energy consumption measurements (in Joules) collected from the experiment. The analysis is made for the client and server sides. To do that, we calculate the average values of the energy consumption collected from the 30 iterations executed per each query and per each strategy (see columns 3 and 7 of Table 3 and Table 4). We then calculate the relative standard deviation of the energy consumption of the previous 30 iterations (see columns 4 and 8). These relative standard deviations show that the fluctuation of values are small (*i.e.,* less than 1) so the average values can be used as representative values for the 30 iterations.

Step 2: we next analyze if there is any significant difference of the energy consumption measurements between the execution of the 22 queries by applying a database optimization strategy, *i.e., index*, *com-*

Table 3: The distribution of the energy consumed by the queries Q01 to Q11. Where: *B = Base*, *Index = Index*, *C = Compression*, *IC = Index and Compression*, *AEC = average of energy consumption (EC in Joule unit)*, *RSEC = relative standard deviation of EC*, *DEC = difference of EC vs Base*, *W= Wilcoxon values vs Base*.

		Client				**Server**			
		AEC	**RSEC**	**%DEC**	**W, *p-value***	**AEC**	**RDEC**	**%DEC**	**W, *p-value***
Q01	B	25.28	0.12	–	–	320.26	0.03	–	–
	I	22.34	0.10	11.63	**200, = 0.0001**	139.65	0.07	**56.39**	**0, < 2.2e-16**
	C	24.04	0.07	–	344, = 0.1194	318.36	0.01	–	454, = 0.959
	IC	22.74	0.08	10.04	**213, = 0.0003**	135.74	0.05	**57.61**	**0, < 2.2e-16**
Q02	B	19.71	0.22	–	–	1.84	0.45	–	–
	I	16.45	0.11	16.48	**242, = 0.0018**	7.62	0.10	-313.5	**900, < 2.2e-16**
	C	18.39	0.12	–	423, = 0.6973	8.38	0.11	-355	**900, < 2.2e-16**
	IC	18.17	0.1	–	437, = 0.8545	7.49	0.11	-306.5	**900, < 2.2e-16**
Q03	B	22.09	0.14	–	–	60.22	0.07	–	–
	I	19.50	0.09	11.74	**232, = 0.001**	57.52	0.04	4.48	**251, = 0.0029**
	C	19.80	0.11	10.4	**269, = 0.007**	53.62	0.04	10.96	**18, = 2.7e-14**
	IC	18.57	0.08	15.95	**132, = 6.2e-07**	54.48	0.05	9.52	**70, = 4.8e-10**
Q04	B	18.95	0.09	–	–	6.15	0.24	–	–
	I	16.62	0.10	12.31	**109, = 1.1e-05**	3	0.19	**51.23**	**2, = 1.6e-14**
	C	16.34	0.10	13.76	**96, = 2.6e-06**	4.25	0.29	30.91	**125, = 5.01e-05**
	IC	16.44	0.08	13.23	**76, = 2.3e-07**	2.76	0.22	**55.06**	**2, = 1.6e-14**
Q05	B	18.78	0.13	-	-	21.43	0.06	–	–
	I	18.32	0.11	–	387, = 0.3581	26.50	0.07	-23.68	**892, = 1.13e-15**
	C	18.12	0.12	–	387, = 0.3581	18.18	0.1	15.13	**53, = 3.57e-11**
	IC	19.14	0.11		486, = 0.6022	25.04	0.08	-16.84	**860, = 3.63e-12**
Q06	B	19.58	0.09	–	–	35.49	0.05	–	–
	I	18.94	0.11	–	342, = 0.1124	34.72	0.06	–	347, = 0.1304
	C	18.30	0.08	6.51	**268, = 0.0066**	33.62	0.06	5.28	**222, = 0.0006**
	IC	18.66	0.11	4.72	**308, = 0.0358**	36.14	0.06	–	533, = 0.2244
Q07	B	18.24	0.14	–	–	28.87	0.07	–	–
	I	18.47	0.10	–	494, = 0.5229	33.82	0.07	-17.12	**854, = 1.08e-11**
	C	18.64	0.10	–	538, = 0.1973	30.58	0.07	-5.88	**646, = 0.0034**
	IC	17.91	0.14	–	412, = 0.5819	33.22	0.08	-15.02	**844, = 5.8e-11**
Q08	B	18.32	0.12	–	–	30.89	0.05	–	–
	I	17.78	0.09	–	381, = 0.3136	39.43	0.04	-27.66	**900, < 2.2e-16**
	C	18.58	0.11	–	478, = 0.6865	29.42	0.05	4.73	**245, = 0.0021**
	IC	18.81	0.11	–	498, = 0.4853	38.19	0.04	-23.63	**900, < 2.2e-16**
Q09	B	18.98	0.13	–	–	60.02	0.03	–	–
	I	16.26	0.11	14.35	**169, = 1.52e-05**	9.23	0.10	**84.62**	**0, < 2.2e-16**
	C	19.40	0.09	–	506, = 0.4147	56.66	0.03	5.6	**72, = 6.3e-10**
	IC	16.61	0.12	12.46	**215, = 0.00038**	4.29	0.18	**92.85**	**0, < 2.2e-16**
Q10	B	19.45	0.08	–	–	57.76	0.03	–	–
	I	18.95	0.12	–	409, = 0.552	23.28	0.06	**59.69**	**0, < 2.2e-16**
	C	19.48	0.1	–	457, = 0.924	56.05	0.03	2.95	**205, = 0.0002**
	IC	18.50	0.1	4.86	**309, = 0.03713**	52.86	0.03	8.48	**20, = 4.59e-14**
Q11	B	17.47	0.06	–	–	4.55	0.16	–	–
	I	15.38	0.10	11.99	**131, = 5.63e-07**	1.98	0.31	**56.51**	**1, < 2.2e-16**
	C	16.74	0.09	–	321, = 0.0853	4.82	0.14	–	509, = 0.2673
	IC	14.79	0.11	15.36	**74, = 8.32e-10**	1.36	0.4	**70.19**	**0, < 2.2e-16**

pression, *index-compression* strategies, versus by not using any optimization strategy, *i.e., base* strategy. To do that, we use the `Wilcoxon` statistical test (see columns 6 and 10). As we observe, not for all the cases a significant difference is reported. For instance, for query 1 (Q01), a significant difference of the energy consumed by the client and server was found for the *index* strategy versus the *base* (see row 4), however for the same query, the difference of energy consumption between the *compression* strategy and the *base* for the client is not significant (see row 5).

Step 3: only for the cases in which a significance difference was obtained in the previous step, we calculate the percentage in which extent the energy consumption measurements were increased or decreased (see columns 5 and 9). To do that, we consider that the average values of the energy consumed by the *base* represent the 100% and we calculate which is the percentage that represent the average values of energy consumed by the

Table 4: The distribution of the energy consumed by the queries Q12 to Q22. Where: *B = Base*, *Index = Index*, *C = Compression*, *IC = Index and Compression*, *AEC = average of energy consumption (EC in Joule unit)*, *RSEC = relative standard deviation of EC*, *DEC = difference of EC vs Base*, *W= Wilcoxon values vs Base*.

		Client				**Server**			
		AEC	**RSEC**	**%DEC**	**W, *p-value***	**AEC**	**RSEC**	**%DEC**	**W, *p-value***
Q12	B	19.22	0.1	–	–	56.08	0.03	–	–
	I	18.07	0.12	6.0	**309, =0.0371**	17.63	0.08	**68.56**	**0, <2.2e-16**
	C	19.44	0.1	–	481, =0.6543	53.33	0.04	4.9	**149, =2.9e-06**
	IC	15.18	0.15	21.04	**88, =5.2e-09**	11.74	0.1	**79.06**	**0, <2.2e-16**
Q13	B	19.12	0.09	–	–	49.71	0.04	–	–
	I	19.02	0.12	–	435, =0.8315	48.05	0.04	3.33	**248, =0.0025**
	C	18.75	0.1	–	395, =0.4232	48.63	0.04	–	328, =0.0723
	IC	17.94	0.11	6.17	**284, =0.0136**	48.38	0.05	2.67	**306, =0.0332**
Q14	B	18.99	0.12	-	-	35.42	0.06	–	–
	I	17.14	0.12	9.75	**255, =0.0035**	7.30	0.27	**79.37**	**0, <2.2e-16**
	C	18.44	0.09	–	379, =0.2996	32.34	0.07	8.71	**149, =2.9e-06**
	IC	16.74	0.09	11.89	**198, =0.00012**	7.32	0.22	**79.33**	**0, <2.2e-16**
Q15	B	26.16	0.21	–	–	98.10	0.16	–	–
	I	18.6	0.11	28.91	**127, =3.8e-07**	46.70	0.07	**52.4**	**0, <2.2e-16**
	C	19.80	0.10	24.31	**185, =5.1e-05**	73.51	0.03	25.07	**46, =1.1e-11**
	IC	18.19	0.10	30.48	**103, =3.1e-08**	47.26	0.06	**51.83**	**0, <2.2e-16**
Q16	B	25.01	0.06	–	–	24.76	0.06	–	–
	I	16.7	0.09	33.24	**0, <2.2e-16**	17.14	0.12	30.78	**0, <2.2e-16**
	C	17.02	0.1	31.94	**0, <2.2e-16**	18.5	0.08	25.28	**1, <2.2e-16**
	IC	15.38	0.09	38.53	**0, <2.2e-16**	17.9	0.05	27.71	**0, <2.2e-16**
Q17	B	43.78	0.08	–	–	393.82	0.03	–	–
	I	22.74	0.09	48.06	**0, <2.2e-16**	241.73	0.01	38.62	**0, <2.2e-16**
	C	23.76	0.08	45.73	**0, <2.2e-16**	265.73	0.02	32.53	**0, <2.2e-16**
	IC	23.39	0.10	46.56	**0, <2.2e-16**	238.51	0.02	39.44	**0, <2.2e-16**
Q18	B	38.64	0.21	–	–	334.93	0.09	–	–
	I	23.57	0.08	39.0	**75, =9.5e-10**	263.12	0.02	21.44	**0, <2.2e-16**
	C	23.13	0.08	40.14	**61, =1.3e-10**	277.66	0.01	17.1	**95, =1.2e-08**
	IC	22.89	0.08	40.77	**56, =5.8e-11**	275.38	0.01	17.78	**49, =1.8e-11**
Q19	B	19.45	0.10	–	–	54.97	0.03	–	–
	I	15.84	0.08	18.54	**43, =6.3e-12**	0.68	0.41	**98.76**	**0, <2.2e-16**
	C	19.64	0.11	–	476, =0.7082	53.76	0.04	2.20	**265, =0.0058**
	IC	15.80	0.09	18.74	**41, =4.4e-12**	0.6	0.43	**98.91**	**0, <2.2e-16**
Q20	B	20.02	0.10	–	–	65.6	0.07	–	–
	I	20.55	0.10	–	519, =0.3136	60.43	0.03	7.88	**99, =1.94e-08**
	C	20.82	0.07	-4.04	**583, =0.0496**	63.94	0.07	–	340, =0.1058
	IC	20.67	0.11	–	522, =0.2928	61.46	0.04	6.3	**172, =1.9e-05**
Q21	B	20.00	0.11	–	–	61.87	0.03	–	–
	I	20.01	0.11	–	447, =0.9707	61.91	0.03	–	443, =0.924
	C	19.86	0.10	–	433, =0.8087	60.32	0.02	2.51	**259, =0.004**
	IC	20.77	0.09	–	527, =0.2601	61.67	0.04	–	407, =0.532
Q22	B	119.39	0.06	–	–	4427.2	0.07	–	–
	I	16.22	0.07	**86.41**	**0, <2.2e-16**	1.41	0.19	**99.97**	**0, <2.2e-16**
	C	118.93	0.06	–	389, =0.7938	4387.18	0.07	–	376, =0.6402
	IC	16.26	0.05	**86.38**	**0, < 2.2e-16**	1.2	0.19	**99.97**	**0, < 2.2e-16**

other strategies. As observed, the majority of these percentages show that a decreasing of the energy consumed by the execution of the queries is produced when a strategy is applied. However, for five queries (see Q02, Q05, Q07, Q08 and Q20), an increasing of the energy consumption is reported when a strategy is applied. Analysing these queries we did not find any particularity that can justify this effect, so more studies are needed in order to explain this phenomena.

RQ1: Does the Implementation of Database Optimization Strategies Affect the Energy Consumption of Client-Server Applications?

From Table 3 and Table 4, we observe that the strategies *index* and *compression* are able to have an impact of the execution of the queries. These impacts are more evident for the server side than for the client side for the majority of the cases.

The *Index* Strategy: we observe that for the majority of queries the *index* strategy is able to impact the en-

ergy consumption of the execution of queries: for four queries (Q02, Q05, Q07 and Q08) the energy consumption is increased, for two queries no significance difference was determined (Q06 and Q21), and for the other 16 queries the energy consumption is decreased by applying the *index* strategy. Regarding the latest 16 queries, the *index* strategy can reduce: i) for the client, at least 6% equivalent to 1.15 J (Q12) and up to 86.41% equivalent to 103.17 J (Q22); ii) for the server, at least 3.33% equivalent to 1.66 J (Q13) and up to 99.97% equivalent to 4425.79 J (Q22).

The *Compression* Strategy: we observe that for the majority of queries the *compression* strategy is able to impact the energy consumption of the execution of queries: for three queries (Q02, Q07 and Q20) the energy consumption is increased, for four queries no significance difference was determined (Q01, Q11, Q13, and Q22), and for the other 15 queries the energy consumption is decreased by applying the *compression* strategy. Regarding the latest 15 queries, the *compression* strategy can reduce: i) for the client, at least 6.51% equivalent to 1.28 J (Q06) and up to 45.73 % equivalent to 20.02 J (Q17); ii) for the server, at least 2.2% equivalent to 1.21 J (Q19) and up to 32.53% equivalent to 128.09 J (Q17).

From previous results, we note that the positive impact is more important by applying the *index* strategy than applying the *compression* strategy for the majority of the queries.

In response to RQ1. Seeing the results, the reduction of the energy consumption for the *index* strategy was between [1.15, 103.17] J for the client side, whilst for the *compression* was [1.28, 20.02] J. For the server side, the *index* strategy was able to reduce between [1.66, 4425.79] J, whilst the *compression* reduced between [1.21, 128.08] J. Therefore, we can conclude that the *index* strategy is more effective than the *compression* strategy in reducing the energy consumed by the execution of SQL queries for relational databases such as the PostgreSQL database system. The reason for this may be due to the mechanism PostgreSQL uses for compression that avoids data repetitions to save space and adds references to the original data having to use them later to recover the original data.

RQ2: Does the Addition of Database Optimization Strategies Affect the Energy Consumption of Client-Server Applications?

The interaction of database optimization strategies is evaluated by applying both strategies (*i.e., index* and *compression*) at the same time. The statistical evaluation for this interaction is also reported in Table 3 and Table 4.

The *Index-Compression* Strategy: we observe that for the majority of queries the *index-compression* strategy is able to impact the energy consumption of the execution of queries: for four queries (Q02, Q05, Q07 and Q08) the energy consumption is increased and we observe that these queries are the same reported for the *index* strategy, so it could indicate that the *index* strategy influence more the interaction *index-compression* than the strategy *compression*. Moreover, for one query no significance difference was determined (Q21), the Q21 also reported a not significance value for *index*, however Q06 that was also reported for the *index* becomes having a significant difference for the interaction *index-compression* for the client. For the other 17 queries the energy consumption is decreased by applying the *index-compression* strategy. Regarding the last 17 queries, the *index-compression* strategy can reduce: i) for the client, at least 4.72% equivalent to 0.92 J (Q06) and up to 86.38 % equivalent to 103.13 J (Q22); ii) for the server, at least 2.67% equivalent to 1.33 J (Q13) and up to 99.97% equivalent to 4426 J (Q22).

From these experiments, we observe that the results of *index-compression* are influenced for both strategies. However as the *index* strategy is more effective than the *compression* in reducing energy consumption as was reported in RQ1, we note that there is a tendency that the strategy *index-compression* is more influenced by the *index* strategy.

In response to RQ2. Seeing the results, the reduction of the energy consumption for the *index-compression* strategy was between [0.92, 86.38] J for the client side, whilst for the server side was between [1.33, 4426] J. These values are very similar to the values obtained for the *index* strategy. Therefore, we can conclude that the interaction *index-compression* is more influenced by the the *index* strategy than the *compression* strategy. The reason could be that the fact of the *index* strategy outperforms the *compression* strategy in reducing the energy consumption for the majority of the cases as was reported in RQ1. Moreover, the reduction of energy consumption is not so different by applying just the strategy *index*, hence we can recommend that use only the strategy *index*, and not the combination of the strategies *index* and *compression*, to reduce the energy consumption of relational databases. It can save effort in applying database optimization strategies for reducing energy consumption of relational databases such as the PostgreSQL database system.

6 THREATS TO VALIDITY

Internal Validity. For collecting energy consumption measurements, we used Juliet, a tool created for our experiment. It could introduce bias, so the building of Juliet, which was inspired by similar tools, was independent of the experiment in order to avoid any bias.
Construct Validity. For the design of the experiment, we use the same computer for running the client and server, it limits our experiment to a monolithic scenario. This threat is partially reduced by Juliet being able to collect the energy consumed by isolated processes, *i.e.,* the processes for the client and server. We plan to run experiments in a real distributed scenario. On the other hand, the fluctuation of energy measurements can affect the results and conclusions of the experiment. To reduce this threat, we run 30 times per each database strategy optimization and query, then we calculate the average values of them. Moreover, the database server can be altered by the execution of previous queries affecting the results and conclusions of the experiment. To reduce this threat, we restart the server in order to collect energy by considering equal conditions. A similar decision is taken for the condition of the database itself each time that a strategy should be implemented, *i.e.,* the database is deleted and recreated with the selected strategy to avoid any alterations in results and conclusions.
External Validity. As our experiment was performed on a specific benchmarking limited to 22 specific queries, our results cannot be generalized. This threat is partially reduced by using the TPC-H benchmarking that is well known in this kind of experiment.

7 CONCLUSION

To better understand the impact of end-to-end query optimization strategies (i.e., *indexation*, *data compression*, and their combination) on the power consumption of RDBMS, from both client and server perspectives, we execute tests, using TPC-H benchmark, configured with 22 queries on a 1GB dataset on PostgreSQL RDBMS. To do so, we propose a monitoring tool, called Juliet, able to monitor and estimate the energy and power consumption of processes executing in Linux-based systems. From the experimental results, we conclude that *indexation* is more effective than *data compression* to reduce the energy consumed by the execution of the majority of the 22 queries. However, more experiments are needed to accurately evaluate the impact on energy consumption and obtain stronger conclusions.

We are working on improving the scenario of the experiments to reflect more real-world deployments (*e.g.,* run client and server in different machines), on classifying the queries according their complexity, responses time and size, and on considering consumption on other resources, such as CPU, I/O usage, and memory. We also plan to apply the same empirical evaluation over other RDBMS and NoSQL datasets.

ACKNOWLEDGEMENTS

This work was partially supported by the "GreenSE4IoT: Towards Energy-efficient Software for Distributed Systems" project whose code is STIC-AMSUD 22-STIC-04, and was partially carried out at the Energy4Climate Interdisciplinary Center (E4C) of IP Paris and École des Ponts ParisTech, which is in part supported by 3rd Programme d'Investissements d'Avenir [ANR-18-EUR-0006-02].

REFERENCES

Guo, B., Yu, J., Yang, D., Leng, H., and Liao, B. (2022). Energy-efficient database systems: A systematic survey. *ACM Computing Surveys*, 55(6):1–53.

Guo, Y., Li, J., Li, X., Li, J., and Li, X. (2017). A green framework for dbms based on energy-aware query optimization and energy-efficient query processing. *Journal of Network and Computer Applications*, 84:118–130.

Karyakin, A. and Salem, K. (2017). An analysis of memory power consumption in database systems. In *Workshop on Data Management on New Hardware*, pages 1–9.

Koçak, S. A., Alptekin, G. I., Miranskyy, A. V., Bener, A., and Cialini, E. (2018). An empirical evaluation of database software features on energy consumption. In *Internat. Conf. on Information and Communication Technology for Sustainability*, volume 52, pages 1–19.

Lella, H. S., Manasa, K., Chattaraj, R., and Chimalakonda, S. (2023). DBJoules: An Energy Measurement Tool for Database Management Systems. *arXiv preprint arXiv:2311.08961*.

Liu, X., Wang, J., Wang, H., and Gao, H. (2013). Generating power-efficient query execution plan. In *International Conf. on Advances in Computer Science and Engineering*, pages 286–290.

Mahajan, D., Blakeney, C., and Zong, Z. (2019). Improving the energy efficiency of relational and NoSQL databases via query optimizations. *Sustainable Computing: Informatics and Systems*, 22:120–133.

Martins, P., Tomé, P., Wanzeller, C., Sá, F., and Abbasi, M. (2021). Comparing oracle and postgresql, performance and optimization. In *Trends and App. in Information Systems and Technologies*, pages 481–490.

Procaccianti, G., Lago, P., and Diesveld, W. (2016). Energy Efficiency of ORM Approaches: an Empirical

Evaluation. In *International Symposium on Empirical Software Engineering and Measurement*, pages 1–10.

Rodríguez, M., Jabba, D., Zurek, E. E., Salazar, A., Wightmam, P., Barros, A., and Nieto, W. (2013). Analyzing power and energy consumption of large join queries in database systems. In *Symposium on Industrial Electronics & Applications*, pages 22–25.

Roukh, A., Bellatreche, L., and Ordonez, C. (2016). Enerquery: Energy-aware query processing. In *International Conf. on Information and Knowledge Management*, pages 2465–2468.

Stavros, H. (2009). Energy efficiency: The new holy grail of data management systems research. In *Biennial Conf. on Innovative Data Systems Research*, pages 1–8.

Xu, Z., Tu, Y.-C., and Wang, X. (2015). Online Energy Estimation of Relational Operations in Database Systems. *IEEE transactions on computers*, 64(11):3223–3236.

Yang, D., Yu, J., He, Z., Li, P., and Du, X. (2023). Applying self-powered sensor and support vector machine in load energy consumption modeling and prediction of relational database. *Scientific Reports*, 13(19097):1–16.

You, X., Lv, X., Zhao, Z., Han, J., and Ren, X. (2020). A survey and taxonomy on energy-aware data management strategies in cloud environment. *IEEE Access*, 8:94279–94293.