Evaluating the energy efficiency of OLTP operations

A case study on PostgreSQL

Raik Niemann[1,2], Nikolaos Korfiatis[2], Roberto Zicari[2], and Richard Göbel[1]

[1] Institute of Information Systems, University of Applied Science Hof, Hof (Germany)
[2] Chair for Database and Information Systems, Institute for Informatics and Mathematics, Goethe University Frankfurt, Frankfurt am Main (Germany)

Abstract. With the continuous increase of online services as well as energy costs, energy consumption becomes a significant cost factor for the evaluation of data center operations. A significant contributor to that is the performance of database servers which are found to constitute the backbone of online services. From a software approach, while a set of novel data management technologies appear in the market e.g. key-value based or in-memory databases, classic relational database management systems (RDBMS) are still widely used. In addition from a hardware perspective, the majority of database servers is still using standard magnetic hard drives (HDDs) instead of solid state drives (SSDs) due to lower cost of storage per gigabyte, disregarding the performance boost that might be given due to high cost.
In this study we focus on a software based assessment of the energy consumption of a database server by running three different and complete database workloads namely TCP-H, Star Schema Benchmark -SSB as well a modified benchmark we have derived for this study called W22. We profile the energy distribution among the most important server components and by using different resource allocation we assess the energy consumption of a typical open source RDBMS (*PostgreSQL*) on a standard server in relation with its performance (measured by query time). Results confirm the well-known fact that even for complete workloads, optimization of the RDBMS results to lower energy consumption.

1 Introduction

A general assumption is that efficient programs are also energy efficient [8]. This seems to be reasonable since a program requiring less computation time will probably also require less energy for its execution. This can be true if we assume a constant energy use of a computer, and the computer performs more tasks in the same amount of time because of time efficient programs.

However, a more detailed analysis may show a different picture. For example it is a well-known fact that one complexity measure like time can be optimized at the costs of another complexity measure of space. An optimization following this path may require that the increased amount of data for a program may be

transferred from one type of memory to a different type of memory. In a modern computer this might result that the data may be moved from the CPU cache to the main memory or from main memory to a drive. An obvious consequence of this transfer are longer access times which may already reduce the benefit of these optimization strategies. In addition the usage of a different memory type may result in higher energy costs [14]. For example, this can be the case if a large data structure needs to be generated on an external drive and the access to this drive would need to be frequently used in comparison to a solution where the drive may be even stopped to save energy.

Since the amount of information to be processed has drastically increased over the last couple of years, large and very large data centers which are expected to support internet operation have been built by major players (e.g. *Google* and *Amazon*). However, increased costs of operation (electricity, cooling costs) have made these vendors to offer the rental of unneeded data center capacity (disk storage space, computation time) to nearly everyone who is willing to pay. This gives rise to the concept of "*cloud computing*" where, for example, server capacities are only needed for a limited time or where the acquisition of an own data center is too expensive [2].

Furthermore, in the context of database applications another issue might arise: the distribution of data across a database cluster as a basis for distributed computing, as it happens in the case of Big Data/*Hadoop* clusters [1]. Industry trends[3] advocate that increased demand for server performance will be fulfilled by deploying more (database) servers and data centers. But with increased energy costs, data centers operators are looking for mechanisms to avoid or reduce those costs (scale-up strategies). This way the energy efficiency of a single database server becomes important as it can be a crucial component for the overall cost assessment of a data center operation where energy costs are the most significant factor on their operation and scalability.

1.1 Background and Motivation

Several approaches which can be found in the literature have addressed the issue of energy efficiency of database servers from different aspects. *Harizopoulos et. al* [13] as well as *Graefe* [5] divided energy efficiency improvements into a hardware and a software part. Recent work such as the one by *Baroso* and *Hölzle* [3] has revealed the fact that the impact of hardware improvements on the energy efficiency is quite low. A main approach in that direction is the use of Solid State Drives (SSD's) which have been shown to improve performance on typical database operations such as sorting and therefore energy efficiency [4]. Obviously SSDs are faster in such kind of scenarios [10], but there are other tradeoffs as a replacement for HDDs with most obvious the cost per gigabyte (*Schall et. all* [11], *Schröder-Preikschat et. all* [12], *Härder et. all* [6]).

On the other side, software improvements seem to be more attractive when it comes to increasing the energy efficiency of a database server, especially for

[3] *State of the data center 2011* by *Emerson*

relational databases. *Lang* and *Patel* [16], for example, proposed a database query reordering technique to influence the energy consumption of the database server. *Xu et. all* [20] proposed a modification of the query planner in order to take the estimated energy consumption into consideration.

Tsirogiannis et. all [14] had a very detailed investigation regarding the relation between performance and energy consumption in a RDBMS. For that they did extensive measurements with various hardware configurations that are typically found in a database server for a scale-out scenario. In addition to this, they identified performance tradeoffs for different SQL query operators.

1.2 Objective of This Study

This study provides the basis for a more detailed analysis of the energy consumption in the context of database applications that are common in daily workloads of enterprise users. For this purpose the paper does not only consider the total power consumption of the full system but also the usage of energy by different components. In addition the paper analyses standard energy saving strategies coming with a modern computer system and their impact on the power consumption.

In the context of database applications, it is important to know which database operation or which combination of SQL operators affects which of the measured components. This study analyzes these combinations and their fraction of the overall power consumption.

All the mentioned aspects lead to the final view on the energy efficiency of a single database server. This is important when it comes to a scale-out scenario that is typically found in a database cluster. There are two choices: either one decreases the energy consumption of the used hardware equipment with a small decrease of the database response time, or increases the performance while accepting a slight increase of the energy consumption. Both ways improve the energy efficiency.

As suggested by *Xu* [18], more experimental research has to be done concerning the energy consumption of database servers. Taking this into account, this study tries to have a closer look on both choices mentioned above and to give recommendations how to improve the energy efficiency of a single database server in a general way (for example the usage of buffers and indices combined with traditional HDDs as the primary storage for the database files).

2 Measurement Methodology

2.1 Test Server Preparation

For the purpose of this study, we constructed a database server making use of recent technical core components. The operating system selected for testing was a typical *Linux* distribution (*Ubuntu Server* version 11.10) using a stable kernel (kernel version: 3.0.0.17). In order to eliminate biases from the operating system

in our measurements, all unnecessary operating system services were turned off. The database management system (DBMS) that was used was *PostgreSQL* version 9.1. The hardware characteristics of the test server are provided in Table 1.

Table 1. Hardware characteristics of the test server used in our study

CPU	*Intel Core i7-860 @2.8 GHz*
Main memory	3x *Samsung* DDR-2 2 GByte 800 MHz (m378b5673fh0-ch9)
Hard drives	3x *Hitachi* 1 TByte, 16 MByte cache (HDT721010SLA360)

Two of the three hard drives were combined as a striping RAID array in order to boost performance and to separate the filesystem calls of the DBMS from the operating system. The operating system itself was installed on the remaining third hard drive.

We identified the core components we were interested in their power consumption throughout the various tests as follows: (a) CPU, (b) main memory, (c) motherboard as a whole and (d) the hard drives of the RAID array. These core components were used as a unity of analysis in order to assess the optimization options. Additionally we were interested in the impact of the operating system settings as well as the test server capabilities on the energy consumption of the core components. Modern server configurations don't allow an external measurement of the energy consumption of internal server components so we decided to use a test apparatus based on moderate hardware. Our test arrangement makes use of a modified ATX power for measurements but server manufactors mostly use proprietary power cords. The complete test arrangement can be viewed online[4].

Besides, the test server's motherboard used *Intel*'s EIST[5]. In general *EIST* enables and disables CPU cores or reduces and raises the overall CPU clock frequency as a function of the CPU usage. This also affects other technical aspects, e.g. the heat dissipation from the CPU.

Recent Linux kernels offer several modules providing more data to the frequency scaling heuristics. The observed modules for our test server configuration are the *on-demand* and the *power-saving* modules. Taking this into account, the command line tool *powertop*[6] was used to suggest configurations on the disablement of operating system services that might influence energy consumption.

[4] Refer to `http://team.iisys.de/test-apparatus/`

[5] Acrynym for *Intel Enhanced Speedstep*. It allows the system to dynamically adjust the CPU frequency and voltage to decrease energy consumption and heat production.

[6] Refer to `http://www.lesswatts.org/projects/powertop/`

2.2 Stress Tests on Energy Consumption

Before assessing the energy efficiency as a whole as well as the components we were interested in profiling their energy consumption, we performed stress tests to get a detailed overview of the energy distribution among the single components. Figure 1 summarizes the stress tests. Each tuple of the shown bars represents a stress test for a core component. The left bar of a specific tuple illustrates the energy consumption of each component in conjunction with the *on-demand* frequency scaling module. The right bar of the tuple displays the energy consumption of all energy saving settings enabled, respectively. Figure 1 also shows the energy consumption of the power supply unit (PSU) which is the difference between the overall power consumption of the test server and the one of the measured components.

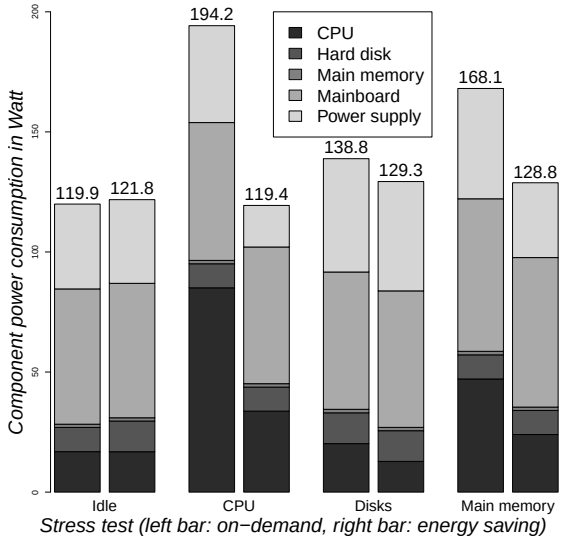


Fig. 1. Energy consumption per component during our calibration stress test

Our first action was to calibrate the test procedure by analyzing the energy consumption of the core hardware components as a basis for the other tests. All services and applications not required for the operating system as well as the DBMS service were turned off. The operating system uses default settings, e.g. the usage of the *on-demand* CPU frequency scaling module. The average consumption per core component is shown in Fig. 1 in the left bar of the tuple named *Idle*. The right bar of this tuple shows the energy consumption with all available energy saving settings turned on. There is a slight increase of the power consumption: although the CPU is forced to reach the energy saving idle states, it is constantly interrupted by doing so, for example by administrative

background processes of the operating system. Changing the CPU state costs some effort but results in higher energy consumption.

To measure the power consumption stressing the CPU of our test server, we decided to use the *Linux* command line tool *burnMMX* because it utilizes all parts of the CPU including extension like *MMX* or *AVX*. Based on the eight CPU cores reported to the operating system, we ran our CPU stress test eight times and increased the number of running *burnMMX* instances accordingly. As assumed the power consumption increases up to the four real cores and remains relatively constant if more cores were used due to HT[7]. This behavior corresponds to the CPU tests reported in [14]. The CPU stress test with the highest energy consumption for both scenarios (energy saving settings turned off and on) is depicted in Fig. 1 as the bar tuple labeled *CPU* for comparison with the other stress tests.

To stress test the hard drives in the RAID array we executed the I/O benchmark suite *iozone* while observing the power consumption throughout the different benchmark tests (different access strategies, buffered and un-buffered data access and so on). Our test results are displayed in Fig. 1 in the second bar tupel labeled *Disks*. Our test shows that the impact on the energy consumption is low when all energy savings settings are turned on. However, in contrast to the power consumption of current SSDs [12] the power consumption of the hard drives is two to three times higher.

Finally we stressed the main memory with the command line tool *memtester* that uses different patterns to access the main memory. It also tests for the correctness of the contents by writing, reading and comparing the main memory areas. The effect on the power lane supplying the main memory was not measurable. In contrast to this, the left bar of the tuple named *Main memory* in Fig. 1 indicates an increase of the overall power consumption of 30 Watts in which the CPU is responsible for. Even with all energy saving settings turned on [19], the overall energy consumption was higher than in idle state.

2.3 Measuring Energy Efficiency

We define the performance ratio (P) of a single database query as the unit of time to execute the query in time (t) to obtain the results [15]:

$$P = \frac{1}{t} \tag{1}$$

We then normalize a set of performance values to values between zero and one as follows:

$$P_{normalized,i} = P_i \cdot \frac{1}{max(P)}$$

[7] *HT* is an acronym for *Hyperthreading* by *Intel*. It is used to improve parallelization of computation.

We then define the *energy efficiency* EE as the ratio between the performance P of the database query and the electrical work W executing the query:

$$EE = \frac{P}{W} \tag{2}$$

We then normalize the set of efficiency values to values between zero and one as follows:

$$EE_{normalized,i} = EE_i \cdot \frac{1}{max(EE)}$$

2.4 Selection of Workloads and DBMS Parameters

As aforementioned our intuition here is to examine the effects of executing a database query in relation to the overall power consumption. For example a combination of a not well formed SQL query and an optimized query planner can cause a cascade of operations that consumes a lot of unnecessary power, e.g. sequential scans cause unnecessary hard drive accesses with all its overhead in the operating system (access control, swapping and so on).

According to related work, for example [14] or [3], we identified several settings for our *PostgreSQL* database server we hypothesized they had an important impact on the power consumption. Those settings can be divided into two groups: the first one are the settings for the underlying operating system and for *PostgreSQL* and the second one are settings for the database itself. In detail those settings are:

- The size of the main memory assigned to *PostgreSQL* to operate
- The size of the miscellaneous buffers, e.g. for sorting resulting rows or caching
- The settings for the query planner
- The size of the data in the database
- The session type of executing subsequent queries (single session vs. multi session)
- Combinations of the different SQL operators and functions, for example increasing number of joined tables or the number of result set dimensions to be restricted

To get comparable results we decided to run three complete database benchmark workloads: *TPC-H* [9], the *Star schema benchmark* (SSB [7]) and a third benchmark constructed specifically for this study, we call it the *W22* benchmark. The first two offer a standardized way for comparison whereas the last one is a workload we composed to analyze the behavior of *PostgreSQL* not covered by the previously mentioned workloads.

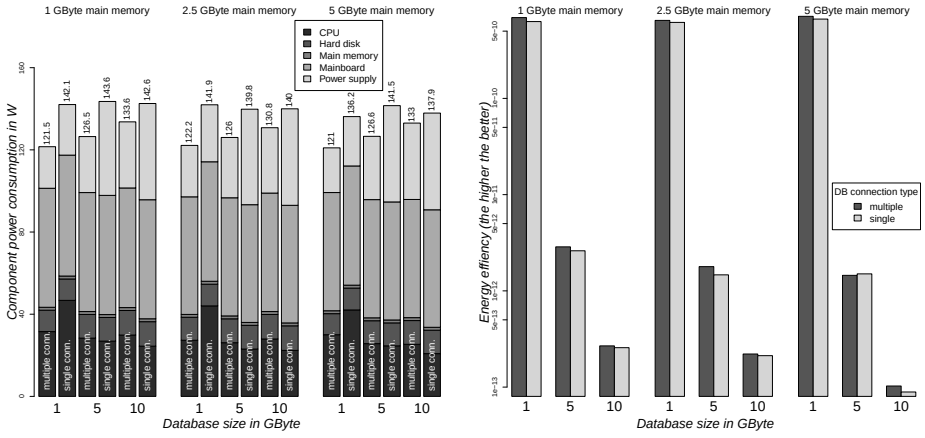
3 Workload Results

3.1 TPC-H Workload

For the TPC-H workload we used their data generator to generate three databases with a size of 1, 5 and 10 GByte of data as well as all suggested indices. We

chose these database sizes because we wanted the databases to fit completely, nearly and under no circumstances in the working main memory for *PostgreSQL* by assigning 1, 2.5 and 5 GByte. This selection was made due to the limitation of 6 GByte overall main memory present in the database server. In a final setup step we optimized the internal data structures and the query planner statistics of *PostgreSQL* for the created databases.

At first we ran the benchmark queries subsequently using a new database connection to avoid *PostgreSQL*'s internal cache[8]. After this we retried the TPC-H benchmark using a single database connection for the SQL queries. The average power consumption per component for both test series is shown in Fig. 2(a).



(a) Energy consumption per component (b) Energy efficiency

Fig. 2. Energy consumption per component and energy efficiency for the TPC-H workload

This figure clearly indicates the higher energy consumption when a single session is used. We recognized that the average overall power consumption for all of our TPC-H tests does not vary a lot. Please compare only the bars which are identically labeled with each other. We assumed that the CPU and hard drives are the main energy consumer but in fact it turned out that the mainboard is the biggest one.

We also assumed that the usage of single database connection would improve the performance because of the better usage of *PostgreSQL*'s internal cache but the opposite occured: the performance was nearly the same with an increased power consumption of 7 percent on average. Using the equations outlined in

[8] Notice that *PostgreSQL*'s implementation isolates client connections completely by running the client sessions in shared-nothing processes. The client processes only share some IPC memory for synchronization purposes and the ACID functionality. This means that every client has its own cache.

Sec. 2.3, this leads to a lower energy efficiency of nearly 8 percent on average as depicted in Fig. 2(b).

A detailed study of the query plans reveals a broad usage of sequential scans on the TPC-H database tables as well as a low usage of the internal caches. Therefore we modified *PostgreSQL*'s settings regarding the query planner and caches to favor the provided indices and repeated the tests. This has an inverted effect: the performance decreases by about 4 percent on average. The reason is the overhead to process the indices which are also disk bounded.

In general we observed a big effect on the energy efficiency by assigning a higher portion of the main memory to *PostgreSQL*. This results in massive swapping for some TPC-H queries, e.g. 1, 9 and 21, and the suspension of the *PostgreSQL* process. In fact, *PostgreSQL* is very disk bound and by accessing the database files the operating system swaps heavily because of the reduced main memory portion. Besides, swapping does effect only the hard drive and not the CPU. This affects the overall power consumption for our TPC-H tests and explains the small variation of the values.

Finally we were interested in performance versus energy efficiency ratio as stated in the conclusion of [14]. This ratio for our entire TPC-H tests with all its different configurations is shown in Fig. 3 and confirms the strong relation between performance and energy efficiency (*the most energy-effiencent configuration is typically the highest performing one*).

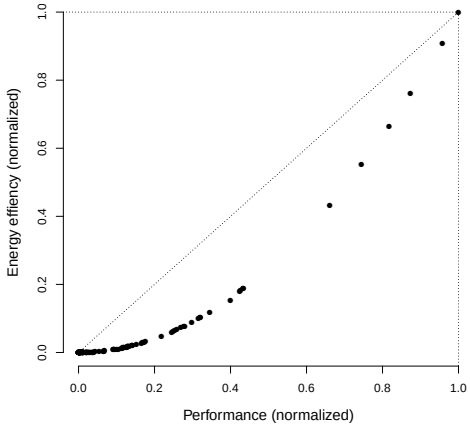


Fig. 3. Performance vs. energy effiency for TPC-H workload

This figure also shows that the majority of the configurations are clustered in the lower left area. This means that most of the queries of the TPC-H benchmark show a poor energy efficiency.

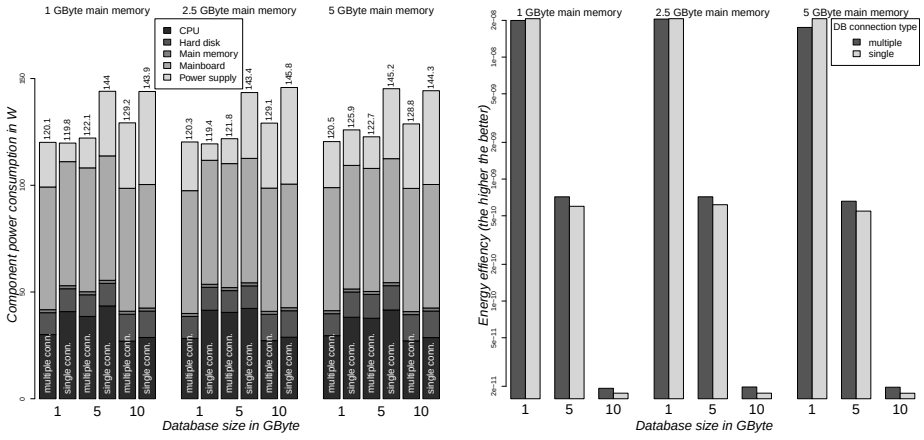
3.2 Star Schema Workload

The *Star schema benchmark* (SSB) was initially composed to get a database layout closer to reality compared to TPC-H which it is based on. According to [7] several original tables were decoupled to make many join operations unnecessary and the set of SQL queries of SSB were created to be more realistic and to test the database capabilities regarding range coverage and indice usage.

For executing the SSB SQL queries we used the same parameters as described in Sec. 3.1: we generated three different databases with 1, 5 and 10 GByte of data and created test configurations for *PostgreSQL* with 1, 2.5 and 5 GByte main memory to work on.

Just as for the TPC-H workload, we ran the SSB database queries subsequently by using a single and multiple database connections. The average energy consumption per component is depicted in Fig. 4(a). The type of accessing the database matters: although the average performance is nearly the same, the average power consumption for multiple connections is lower than the one for using a single connection. The latter consumed nearly 10 percent more energy on average. Except for the database size of 1 GByte, this results in a lower energy efficiency.

In general and in terms of energy efficiency, the SSB benchmark performs better than TPC-H. In addition to this, SSB organizes its SQL queries into groups in which the requested rows do not overlap and the group number indicates on how many dimensions the result set has to be restricted. This allows a better analysis of the results and avoids side effects interfering the results. Figure 4(b) shows the energy efficiency of our SSB tests.



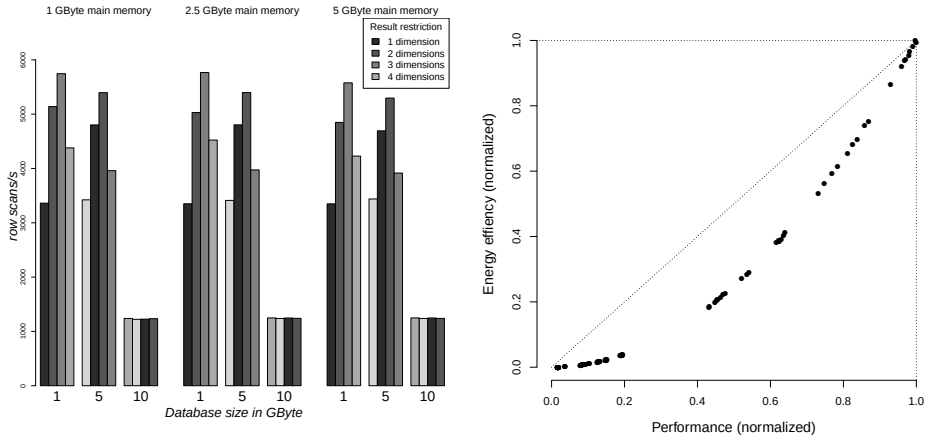
(a) Energy consumption per component (b) Energy efficiency

Fig. 4. Energy consumption per component and energy efficiency for SSB workload

A deeper investigation of the query plans showed us that the reduced set of tables and the modified queries lead to a more stable and predictable behaviour concerning the power consumption and energy efficiency. In particular this can be seen in Fig. 4(b): the energy efficiency for a given constant SSB database size does not vary a lot and is relatively independent from the amount of main memory spent for *PostgreSQL*.

Besides, the energy efficiency of the SSB database with 10 GByte is remarkably lower than the ones with 1 and 5 GByte of data. An investigation of the logged system activities during the tests revealed heavy swapping actions which caused the suspension of the *PostgreSQL* process handling the queries. As illustrated in Fig. 4(a), the low CPU energy consumption indicates the heavy swapping activity.

The query plans also show the general use of sequential scans on the SSB database tables. In fact, no indices were involved to execute the queries in all configurations. The execution of the queries purely relies on the read performance of the database files. As Fig. 5(a) indicates, it does not matter if the result set of the joint tables are further restricted. In addition to this, this figure shows that main memory spent for buffers is relatively unimportant: if one consider a specific database size, the row scan per second rate does not vary a lot. This means that the mentioned rate is independent of the main memory fraction assigned to *PostgreSQL* and also independent of the number of dimensions the result set is restricted.



(a) SSB row scans per second

(b) Performance vs. energy efficiency for SSB

Fig. 5. Row scan count and performance vs. energy efficiency for SSB

The performance vs. energy efficiency ratio for all SSB test configurations is illustrated in Fig. 5(b). The same strong relationship can be seen as in Fig. 3

for the TPC-H benchmark tests. In contrast to TPC-H, the vast majority of the SSB test configurations is clustered in the upper right corner of Fig. 5(b). This means that the SSB configurations perform much better. This leads to a better energy efficiency.

3.3 W22 Workload

To get a more in-detailed look at the impact of the different SQL operations and their combinations we composed a database workload called W22. The workload consists of six groups of database queries:

1. aggregate functions (`count`, `avg` and `sum`)
2. grouping (`GROUP BY`)
3. sorting (`ORDERED BY`)
4. selecting different data types, e.g. `int`, `varchar`, `text` and `date`
5. removing duplicates (`DISTINCT`)
6. joins (cross joins, conditional joins)

In contrast to this, the TPC-H and the SSB workload are designed for benchmark an OLAP scenario. Their database queries use a mix of different SQL operations coming from the groups above. Our W22 workload instead tries to analyze the impact of SQL queries in each group. Besides, this workload allows to analyze the behaviour of *PostgreSQL*'s internal query optimizer and query planner as well as the interaction with the underlaying operating system. The W22 queries operate on the TPC-H databases described in Sec. 3.1.

Aggregate Functions Our first tests with aggregate functions, e.g. `avg()`, `sum()` and `count()`, show in particular the impact of the filesystem cache of the underlaying operating system.

All of the mentioned aggregate functions cause *PostgreSQL* to perform a sequential scan. The first query that was executed (`count(*)`) had a notably longer execution time compared to the next ones (`avg()` and `sum()`). The query plans for all three queries are the same. After a closer investigation of the logged activities of the operating system, we identified the filesystem cache as the performance booster by caching the database files that *PostgreSQL* uses. In this case the kind of executing the queries (single session vs. multi session) does not matter. The most important setting is the fraction of main memory assigned to *PostgreSQL*: the lower the fraction the more main memory is available for system caches.

The suggested optimizations of the queries, e.g. the use of an indexed column for `count()` (`count(column)` instead of `count(*)`), did not change the execution times.

Grouping and Sorting The queries of group 2 (grouping) and group 3 (sorting) expands the `count()` query from group 1 with SQL operators like `ORDER BY` and `GROUP BY` to get comparable results. Those queries were composed to study

the impact of the mentioned SQL operators in reaction of the previous tested benchmarks TCP-H and SSB.

The tests result indicates only a slight difference in terms of execution times and energy consumption compared to the test results of group 1. The impact on the energy consumption of the CPU and the main memory was not measurable. As a result the calculated energy efficiency remains the same.

Selection of Different Data Types According to our tests, the selection of a specific data type, e.g. `VARCHAR`, `DATE` and `INTEGER`, has no impact on the query execution time.

The other purpose of the queries of group 4 was to test the operators of the different data types with and without having an index on the particular table column. We observed a big impact on the execution time when an index was used whereas the average energy consumption was slightly higher. The presence of an index in combination of an operator[9] supporting this index lead to an immense performance gain. Therefore the energy efficiency is quite high compared to a sequential scan. In contrast to the queries with an absent index, the queries with an involved index showed an almost linear performance. This is crucial since we used the largest table of TPC-H, `lineitem`, which also has the greatest amount of rows throughout all queries of this group.

PostgreSQL uses an index for a query when a) the index supports the operator of the query and b) the costs for processing the index are lower than sequentially scanning the table. Those costs are composed of customizable base costs and dynamic cost estimations that *PostgreSQL* gathers periodically and statistically from all tables of a database.

We modified the settings for the base costs of loading and processing a database and an index page (usually 8 KBytes of data) to favor the usage of indices, e.g. if the database and indice files are stored on different storage devices with different access speeds (the indice files are usually stored on the faster one). Our test results remain the same because the data and index files are stored on the same hard drive device.

Joining Tables and Eleminating Duplicates Based on our TPC-H and SSB benchmark results, we were interested in the behaviour of *PostgreSQL* dealing with table joins. There are two kinds of joins supported in *PostgreSQL*: unconditional and conditional joins.

Our first query of this W22 group deals with an unconditional join of two tables where the cross product is further restricted by the conditions given after the `WHERE` clause (one restriction per involved table). We expected this query to be unperformant due to the cross product and the successive restriction of the

[9] For example, the operators greater than, less than and equals (`<`, `>` and `=`) are valid for B-tree indices on numeric table columns in *PostgreSQL*. There are other index types, e.g. inverted index, for other column data types as well as their specialized operators.

result set, but this was not the case. The query plan reveals the (unintentional) use of an index for one restriction and a sequential scan for the other one. So the results (performance, energy consumption and the energy efficiency) are the same as mentioned in the last section although a sequence scan is part of the query plan.

The other queries of this W22 group were composed to join two TPC-H tables using inner[10] and equi-joins[11] to examine differences in the query plans. Interestingly those queries indicate the same characteristics in terms of the query planner. For *PostgreSQL* it does not matter where the condition for joining two tables is placed. In other words, *PostgreSQL* does not distinguish between an inner, implicit or equi-join.

Besides, the queries are composed to join two TCP-H tables with different number of rows to investigate the join performance. The first two queries joined the `lineitem` table with the `orders` and the `part` table, respectivly. The last query joined the `orders` with the `customers` table.

Finally we select the amount of joined rows by using the `count(*)` aggregate function. This forces *PostgreSQL* to use sequential scans for the mentioned tables.

As assumed, our test results indicate that the join performance is strongly related to the row scan performance. The test results are similar to the ones of our TPC-H and SSB benchmarks. This means they do not resemble much in their energy consumption but in their execution times. Unsurprisingly the lower the amount of rows to be scanned for joining, the lower is the execution time and therefore the energy efficiency is quite better. Although compared with query with the unconditional join mentioned above, the performance is fundamentally worse.

We were also interested in the effects of eleminating duplicates from a result set. For this purpose we formed a test query using the `DISTINCT()` clause on a column of the `lineitem` table not having a supporting index. The query plan revealed a sequence scan and the removal of duplicates by hashing the values. Again, the test results showed the same characteristics as all of our database tests performing a sequence scan.

4 Summary

At first, our tests with our database server using normal HDDs indicates an energy increase of roughly 9 W when the HDDs are fully utilized. Compared to the energy consumption of the other measured core components during the tests, this is insignificant. The argument, SSDs should be prefered because they consume up to 12 times less energy compared to HDDs, is invalid in this context.

[10] The SQL standard standardized an inner join as `<table a> [INNER] JOIN <table b> WHERE <a.xyz> = <b.xyz>`.

[11] An equi-join is in the form `<table a> JOIN <table b> ON <a.xyz> = <b.xyz>`. The SQL standard allows shorthand for the column to join by using the `USING` clause.

As *Lang et all.* stated in the summary of [17], evaluating the energy efficiency of a DBMS needs the inclusion of entire workloads, not just single queries. This study makes use of three different and complete workloads that allows a more comprehensive look at the energy efficiency of a relational DBMS. Most of the benchmark queries caused a massive usage of sequential scans. This implies that the sequential read performance is an extremly important factor that affects the energy consumption. Actual SSDs clearly outperform normal HDDs but in this case enterprise grade HDDs can be used because they offer nearly the same performance as SSDs.

As mentioned in the introductionary section of this paper, there are more factors and not only technical parameters that influence the performance and thus the energy efficiency of a database server.

For example, the filesystem cache provided by the operating system is more relevant for the execution of a database query in *PostgreSQL* than its internal cache. Based on our experiments, we recommend not to assign more than 50 percent of the main memory to *PostgreSQL* for operations. With more assigned main memory the remaining processes of the operating system are forced to use the remaining portion. This causes the operating system to swap this portion to the hard drive which leads to a dramatic reduction of the performance.

Another important factor for the energy efficiency of the used database benchmark is the kind of accessing *PostgreSQL* (single vs. multiple database connections). Our assumption, subsequent queries of the benchmarks would benefit from *PostgreSQL*'s internal cache by using just a single database connection, does not come true. In fact, the opposite performed better.

Our tests also indicate the fact that energy saving settings are counterproductive for a database server that is reasonably utilized because it decreases the overall system performance.

References

1. Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D., Silberschatz, A., Rasin, A.: HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads. Proc. VLDB Endow. 2(1), 922–933 (2009), http://dl.acm.org/citation.cfm?id=1687627.1687731
2. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Commun. ACM 53(4), 50–58 (2010), http://doi.acm.org/10.1145/1721654.1721672
3. Barroso, L., Holzle, U.: The Case for Energy-Proportional Computing. Computer 40(12), 33–37 (2007)
4. Beckmann, A., Meyer, U., Sanders, P., Singler, J.: Energy-efficient sorting using solid state disks. In: Green Computing Conference 2010. pp. 191–202 (2010)
5. Graefe, G.: Database servers tailored to improve energy efficiency. In: Proceedings of the 2008 EDBT workshop on Software engineering for tailor-made data management. pp. 24–28. SETMDM '08, ACM, New York and NY and USA (2008), http://doi.acm.org/10.1145/1385486.1385494

6. Härder, T., Hudlet, V., Ou, Y., Schall, D.: Energy efficiency is not enough, energy proportionality is needed! In: Proceedings of the 16th international conference on Database systems for advanced applications. pp. 226–239. DASFAA'11, Springer-Verlag, Berlin, Heidelberg (2011), http://dl.acm.org/citation.cfm?id=1996686.1996716
7. O'Neil, P.E., O'Neil, E.J., Chen, X.: The Star Schema Benchmark (SSB), revision 3 (2007), http://www.cs.umb.edu
8. Papadimitriou, C.H.: Computational complexity. In: Encyclopedia of Computer Science, pp. 260–265. John Wiley and Sons Ltd, Chichester and UK, http://dl.acm.org/citation.cfm?id=1074100.1074233
9. Poess, M., Floyd, C.: New TPC benchmarks for decision support and web commerce. SIGMOD Rec 29(4), 64–71 (2000), http://doi.acm.org/10.1145/369275.369291
10. Polte, M., Simsa, J., Gibson, G.: Comparing performance of solid state devices and mechanical disks. In: Petascale Data Storage Workshop, 2008. PDSW '08. 3rd. pp. 1–7 (2008)
11. Schall, D., Hudlet, V., Härder, T.: Enhancing energy efficiency of database applications using SSDs. In: Proceedings of the Third C* Conference on Computer Science and Software Engineering. pp. 1–9. C3S2E '10, ACM, New York and NY and USA (2010), http://doi.acm.org/10.1145/1822327.1822328
12. Schröder-Preikschat, W., Wilkes, J., Isaacs, R., Narayanan, D., Thereska, E., Donnelly, A., Elnikety, S., Rowstron, A.: Migrating server storage to SSDs. In: Proceedings of the 4th ACM European conference on Computer systems. p. 145. EuroSys '09, ACM, New York and NY and USA (2009)
13. Stavros Harizopoulos, Mehul A. Shah, Justin Meza, Parthasarathy Ranganathan: Energy Efficiency: The New Holy Grail of Data Management Systems Research. In: CIDR'09 (2009)
14. Tsirogiannis, D., Harizopoulos, S., Shah, M.A.: Analyzing the energy efficiency of a database server. In: Proceedings of the 2010 international conference on Management of data. pp. 231–242. SIGMOD '10, ACM, New York and NY and USA (2010)
15. Wang, J., Feng, L., Xue, W., Song, Z.: A survey on energy-efficient data management. SIGMOD Rec 40(2), 17–23 (2011), http://doi.acm.org/10.1145/2034863.2034867
16. Willis Lang, Jignesh M. Patel: Towards Eco-friendly Database Management Systems. In: CIDR'09 (2009)
17. Willis Lang, Stavros Harizopoulos, Jignesh M. Patel, Mehul A. Shah, Tsirogiannis, D.: Towards Energy-Efficient Database Cluster Design. CoRR abs/1208.1933 (2012)
18. Xu, Z.: Building a power-aware database management system. In: Proceedings of the Fourth SIGMOD PhD Workshop on Innovative Database Research. pp. 1–6. IDAR '10, ACM, New York, NY, USA (2010), http://doi.acm.org/10.1145/1811136.1811137
19. Zheng, H., Zhu, Z.: Power and Performance Trade-Offs in Contemporary DRAM System Designs for Multicore Processors. IEEE Transactions on Computers 59(8), 1033–1046 (2010)
20. Zichen Xu, Yi-Cheng Tu, Xiaorui Wang: Exploring power-performance tradeoffs in database systems. In: ICDE'10. pp. 485–496 (2010)