Dieses Dokument ist eine Zweitveröffentlichung (Postprint) / This is a self-archiving document (accepted version):

Thomas Kissinger, Dirk Habich, Wolfgang Lehner

Adaptive Energy-Control for In-Memory Database Systems

Erstveröffentlichung in / First published in:


DOI: [https://doi.org/10.1145/3183713.3183756](https://doi.org/10.1145/3183713.3183756)

Diese Version ist verfügbar / This version is available on:

[https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa2-791673](https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa2-791673)
Adaptive Energy-Control for In-Memory Database Systems

Thomas Kissinger  Dirk Habich  Wolfgang Lehner
Database Systems Group
Technische Universität Dresden
Dresden, Germany
first.last@tu-dresden.de

ABSTRACT
The ever-increasing demand for scalable database systems is limited by their energy consumption, which is one of the major challenges in research today. While existing approaches mainly focused on transaction-oriented disk-based database systems, we are investigating and optimizing the energy consumption and performance of data-oriented scale-up in-memory database systems that make heavy use of the main power consumers, which are processors and main memory. We give an in-depth energy analysis of a current mainstream server system and show that modern processors provide a rich set of energy-control features, but lack the capability of controlling them appropriately, because of missing application-specific knowledge. Thus, we propose the Energy-Control Loop (ECL) as an DBMS-integrated approach for adaptive energy-control on scale-up in-memory database systems that obeys a query latency limit as a soft constraint and actively optimizes energy efficiency and performance of the DBMS. The ECL relies on adaptive workload-dependent energy profiles that are continuously maintained at runtime. In our evaluation, we observed energy savings ranging from 20% to 40% for a real-world load profile.

CCS CONCEPTS
• Information systems → Main memory engines; Relational parallel and distributed DBMSs; Query optimization;

KEYWORDS
in-memory, database systems, energy efficiency, adaptivity

ACM Reference format:
https://doi.org/10.1145/3183713.3183756

1 INTRODUCTION
To keep pace with the ever-increasing data volume, modern state-of-the-art in-memory database systems need to scale up on server hardware featuring an increasing amount of main memory and compute resources. From a hardware perspective, this scalability is achieved by implementing a non-uniform memory access (NUMA) resulting in separate memory domains per processor (also referred to as CPU or socket) – or even inside of a single processor – that are remotely accessible via an interconnect network. Efficient data processing on scale-up systems faces several issues as increased latency and decreased bandwidth when accessing remote memory domains. To further scale up on NUMA systems, database systems need to carefully consider these issues as well as the limited scalability of synchronization primitives such as atomic instructions [11, 12, 21, 23]. Hence, the data-oriented architecture was proposed that turned out to show a superior scalability on common scale-up hardware architectures [12, 18, 20]. In contrast to the traditional transaction-oriented architecture, transactions are not first-class citizens anymore and are processed in a distributed fashion by worker threads operating on their local data partitions respectively data structures. The communication between worker threads is handled via a message passing layer as shown in Figure 1.

In addition, the power draw of server hardware in general and especially database servers as a fundamental component of mostly every service became a severe issue in the recent years. Besides from limiting their further scalability, the resulting energy consumption amounts to a significant and increasing fraction of the worldwide energy draw [4, 25]. Hardware vendors have become more and more aware of this problem and focus on increasing the energy efficiency (performance per Watt) and energy proportionality (proportionality between supplied performance and power draw) of processors and servers as a whole. For instance, while the idle power consumption was more than 50% of the peak power consumption in 2010 [26], today’s servers consume only about 20% (c.f., Section 2),
which is a trend towards energy proportionality. Regarding energy efficiency, the TDP (Thermal Design Power) of CPUs was reduced within the last years while performance still improved, which is a clear indicator for energy efficiency advancements. Besides from architectural improvements and shrunk feature sizes of processors, those advancements were achieved by adding a rich set of energy-control knobs (e.g., sleep states, separate core and uncore clocks). However, these energy-control knobs need to be actively controlled at runtime to balance performance and energy-efficiency.

Nowadays, such energy-control knobs are mainly controlled by the hardware itself and the operating system. However, modern in-memory database systems do all the thread scheduling on their own (e.g., to ensure NUMA awareness) and threads are rarely blocked while waiting on secondary storage devices. Especially in the case of a data-oriented architecture, threads are pinned to specific hardware threads to guarantee a local data access and to avoid starving spinlocks, for instance, when multiple threads share the same hardware thread. Moreover, for performance reasons, the entire messaging system is implemented using a polling-based approach instead of using the integrated signaling facilities of the operating system resulting in an always-on state by default. Hence, hardware and operating system have almost no chance to appropriately configure the energy-related tuning knobs of the hardware due to their limited DBMS insights. Thus, it is an essential step to add energy-capabilities to the database runtime to reduce its energy footprint by increasing energy efficiency as well as energy proportionality. Recent works already addressed the topic of energy efficiency in the context of database systems reaching from the energy analysis of database servers [26] or operators [5, 8, 22, 28] to active energy control either by adding energy as an additional optimization goal to the query optimizer [14, 29] or by leveraging feedback-control loops [13, 27, 30]. Nevertheless, most of the works focused on the combination of disk-based transaction-oriented DBMSs having completely different preconditions for energy optimization mechanisms compared to data-oriented in-memory database systems.

In this paper, we present a novel approach for adaptive energy-control of data-oriented in-memory database systems running on NUMA-based scale-up hardware architectures. As a prerequisite, we modify the data-oriented architecture in a way that we are able to selectively turn off worker threads without losing the access to their assigned data partitions, which has a major impact on the message passing subsystem. Based on that, we present the hierarchical Energy-Control Loop (ECL), which is organized in multiple socket-level ECLs and a system-level ECL. While the first ones are responsible for configuring the socket-local hardware components using an adaptively maintained energy profile, the latter one keeps track of current query latencies and uses a best-effort strategy to stay within a user-defined latency limit as shown in Figure 1. In our evaluation we measured energy savings ranging from 15% to 40% depending on the current workload and system load.

**Contributions.** Our contributions are as follows:

1. We analyze the energy optimization potential as well as the energy tuning knobs of a current mainstream server system and quantify the impact of the respective knobs on performance and power draw. Moreover, we investigate the capabilities of the hardware to control those knobs on its own.

2. We eliminate the shortcomings of the data-oriented architecture such that we are able to elastically adjust the number of worker threads at runtime without loosing the access to certain data partitions. In that context, we present a hierarchical message passing layer that is necessary to support those changes.

3. We propose the concept of low-level high-resolution energy profiles that take query interferences into account and describe how they can efficiently be generated and maintained at runtime. With the help of such profiles, we show how different workload types affect the optimal compute resource configuration in terms of energy efficiency and supplied performance.

4. We propose the Energy-Control Loop (ECL) as a novel approach for adaptive energy-control on data-oriented scale-up in-memory database systems. The ECL saves energy by frequently adapting the compute resource configuration while trying to stay within a given query latency constraint. We give an exhaustive evaluation of the ECL regarding all important aspects including an end-to-end evaluation using a real world load profile.

**Paper organization.** The paper is structured as follows: First, we experimentally evaluate the effects of hardware energy-control knobs of a current mainstream server in Section 2. Afterwards, we discuss the overall architecture of our approach including the required changes to the data-oriented architecture in Section 3. In Section 4, we present our concept of energy profiles as a foundation for the ECL. Subsequently, we explain the functioning of our hierarchical ECL and how it utilizes as well as maintains the energy profiles in Section 5. Afterwards, in Section 6, we give an end-to-end evaluation of our ECL. Finally, Section 7 summarizes the related work and Section 8 concludes the paper.

## 2 ENERGY-CONTROL KNOBS IN CURRENT MAINSTREAM SERVERS

In this section, we analyze the energy optimization potential as well as performance and power characteristics of a current server system. Especially, we focus on the available energy-control features and the energy-related decisions that are made by the CPU itself.

**System Under Test.** Due to the near-total market dominance of Intel, we are investigating a representative 2-socket server-class system equipped with Intel Xeon E5-2690 v3 CPUs (Q3 2014) of the Haswell-EP generation and 256GB DDR4 RAM (8x 32GB PC4-2133 LRDIMMs). Each CPU consists of 12 physical cores resulting in...
24 hardware threads with HyperThreading\(^1\) enabled. Compared to previous generations, the Haswell-EP generation includes a bunch of new energy-control features. One unique feature are the fully integrated voltage regulators (FIVR). While previous generations use a single clock for the whole processor, FIVRs enable independent clocks for individual parts of the CPU that allow a more fine-grained dynamic voltage and frequency scaling (DVFS). Figure 2 shows the available clocks inside our evaluation system. Each physical core has a separate clock that is shared by its hardware threads. Moreover, each processor features a separate uncore clock\(^2\), that affects power and performance of the last-level cache (LLC) and the four memory controllers. Additionally, the CPU implements features such as the energy-efficient turbo (EET) and the energy-performance bias (EPB), which we will investigate in the remainder of this section. To measure the energy consumption of the system we either use an LMG450 power meter that is attached to the power supply unit, or the per-processor integrated RAPL counters \[10\]. RAPL counters are highly accurate on this platform and allow us to separately measure the power consumption of the package domain (cores and caches) and the memory controller domain.

Figure 3: Haswell-EP power breakdown into static and dynamic consumption. RAPL and PSU measurements in Watt.

2.1 Static and Dynamic Power Consumption

With our first experiment, we figure out which components of the system draw which amount of power in idle mode and under full load. To do so, we first measure the power values from the external power meter, which is attached to the power supply unit (PSU), as well as the internal RAPL counters in idle mode (static power draw) and once again under full load (dynamic power draw). To get the system under full load, we use the FIRESTARTER tool \[6\] that uses the optimal balance of compute instructions, AVX instructions, and memory controller requests. The results are visualized in Figure 3. The overhead denominates the power share that can not be captured by RAPL counters. Note that the figure does not include the turbo boost peak of 500 Watts, since this high load can only endure for about 1 s due to thermal limitations. The main conclusions we can draw from this experiment is that the static power consumption of the bare server system is only about 18 % of the peak power, which is a great advancement in terms of energy proportionality compared to the number of over 50 % reported in 2010 \[26\] and thus, opens up a lot of space for energy optimizations. Moreover, we can see that the largest amount of the dynamic power is consumed by CPU and DRAM, which also generates a dynamic power overhead – originating from power conversion loses of the PSU, CPU fans, and the motherboard – of about 15 % that can not be measured via RAPL counters. However, since power meters are commonly not attached to servers, we will stick with the RAPL measurements, which are proven to accurately correlate with the PSU power consumption \[7\]. Hence, relative power and energy differences measured via RAPL are highly accurate, while the absolute numbers omit the overhead.

Figure 4: Power costs for activating cores and HyperThreads. Different core and uncore frequency combinations.

Figure 5: Socket-specific power consumption for different uncore clocks and inter-socket dependencies.

Figure 6: Memory bandwidth and power draw for different core and uncore frequency settings. all cores are active.

2.2 C-States and P-States

Intra-Socket. On modern CPUs, single cores or the entire processor can be power-gated to save energy, if not utilized (C-states). Additionally, the hardware implements power states (P-states), which decrease voltage and frequency to operate in a more energy-efficient state at the cost of reduced performance. One innovation of the Haswell-EP generation is that individual core clocks as well as the...
uncore clock can be set independently (cf., Figure 2). On our test system, core clocks can be set between 1.2 and 2.6 GHz (3.1 GHz TurboBoost) and the uncore clock ranges from 1.2 GHz to 3.0 GHz. In Figure 4, we experimentally evaluated the impact of C-states and P-states on the power consumption for a compute-bound workload (incrementing a thread-local counter) using RAPL on a single socket. The results show that most of the power costs incur when the first core of a socket is activated, while activating an additional core causes a much lower power draw and activating HyperThread siblings results in almost no cost. The high power costs for activating the first core adhere to the uncore clock, which can be halted, if no core is active on the CPU. Thus, we observe a correlation to the uncore clock. Halting the uncore clock allows the processor to power-gate the power-hungry LLC, which saves up to 30 W. The power costs for activating additional physical cores depend on the respective core clock and are almost constant.

Inter-Socket. However, since processors are able to access memory on a remote socket, the uncore clock of a socket can not be halted unless all processors of the system halted their uncore clock, too. Figure 5 shows the power consumption of the individual sockets for a halted uncore clock, which requires that both sockets halted their uncore clock, and for different uncore frequencies, when the other socket is active. The experiment shows the mentioned dependency and also demonstrates that the second socket is consuming less power compared to the first one, especially when the uncore clock of all processors is halted. We were not able to uncover the root cause of this difference between the two sockets.

Memory Bandwidth. In our last experiment, we quantify the impact of core and uncore clocks on the bandwidth. As Figure 2 shows, the memory bandwidth mainly depends on the uncore clock, because it affects the performance of memory controllers and the LLC. Our experiment in Figure 6 confirms this assumption. It shows, that the available memory bandwidth depends on the uncore clock and that nearly full bandwidth can be achieved when operating all of the cores in their lowest available P-state (1.2 GHz) as long as the uncore clock is set to its maximum (3.0 GHz).

2.3 CPU-Driven Energy Management

So far, we analyzed the power and performance implications of the various energy-control features the hardware is offering. In the following experiments, we will focus on the decision quality of the CPU to manage the core and uncore frequencies on its own.

Core Frequency Scaling. The energy-performance bias (EPB) can be set per hardware thread and influences the energy-efficient turbo (EET) as well as the uncore frequency scaling (UFS) [3]. The EPB can be set via a machine-specific register (MSR) to powersave, balanced, or performance. To evaluate the decisions made by the EPB, we start with executing the compute-bound workload on all cores of a processor and change the frequencies of all cores from 1.2 GHz to the maximum frequency (3.1 GHz w/ turbo) for different EPB settings and measure the instructions retired (completed instructions) as well as the power consumption using RAPL. Figure 7(a) shows the behavior over time for the powersave or balanced EPB setting. The frequency change happens at the 1000 ms time point. We observe that the CPU immediately draws more power and sticks for about 1 s (delayed RAPL measurements) at 2.6 GHz and afterwards enters the turbo mode (3.1 GHz), which causes an additional power draw and a significant performance increase. Repeating the same experiment having the EPB set to performance, lets the CPU immediately enter the turbo mode after the frequency change, avoiding the 1 s delay as shown in Figure 7(b). Figure 7(c) visualizes the results for the experiment with an EPB setting of powersave or balanced, this time using a memory-bound workload (scan over an array). Here, we observe the same delay of 1 s before activating the turbo mode. However, the experiment demonstrates that this decision was a bad one in terms of energy efficiency, since the CPU invested a
lot of power, but was not able to increase the performance as the instructions retired measurement indicates. Thus, the only impact of the EPB on the core frequency we could observe, was that a 1 s delay was added before entering the EET.

Uncore Frequency Scaling. To evaluate the impact of the uncore frequency scaling (UFS) decisions, we run the compute-bound workload with all cores running at maximum frequency and compare the measurements of power consumption and performance for automatic UFS as well as for having the uncore frequency pinned to 1.2 GHz or 3 GHz. Figure 8 shows the respective results. The number of retired instructions is the same for all uncore clock settings with a slight advantage for the lowest uncore frequency. Nevertheless, the automatic UFS decides to use the highest uncore frequency, which draws additional 12 W compared to the 1.2 GHz setting, which even delivers a bit more performance. Thus, this experiment once again confirms a bad decision making of the built-in CPU power management facilities and suggests to set the EPB to performance mode, when doing explicit energy control.

Conclusions. Our analysis of the server demonstrates that there are opportunities to save power on current hardware. These power savings can be achieved by appropriately configuring the energy-control knobs of the hardware, which significantly influence performance and power consumption. Moreover, our evaluation also revealed that the decision making of the CPU in terms of power management is not very sophisticated, actually, which motivates explicit energy-control for in-memory database systems.

3 ARCHITECTURAL OVERVIEW

In this section, we give an overview of the overall architecture of our approach. At first, we briefly introduce the basic data-oriented architecture and highlight its energy-related issues. Afterwards, we discuss the necessary elasticity extensions for this architecture and focus on the major architectural implications. Finally, we give an overview of the Energy-Control Loop (ECL) and discuss the interplay with the database system runtime.

Data-Oriented Architecture. The data-oriented architecture was proposed as the antagonist to the traditional transaction-oriented architecture, which is implemented by a majority of the DBMSs. Its goal is to overcome the scalability limitations of the transaction-oriented architecture, which is a prerequisite for energy proportionality, because activating twice of the hardware resources should ideally result in a doubled performance and power draw. The core concept is that all data objects are implicitly partitioned and partitioned data are exclusively accessed by the assigned worker thread that is pinned to a specific hardware thread. The communication between worker threads during query processing is handled via a message passing layer. The data-oriented architecture as it was originally proposed [18] faces several issues in terms of energy consumption:

Static Mapping. Data partitions are statically mapped to specific worker respectively hardware threads. Thus, data partitions become unavailable as soon as worker threads are disabled and the corresponding hardware thread enters a sleep state.

Load Balancing. This static mapping leads to an odd utilization of compute resources, which is challenged by balancing data partitions at runtime. However, data balancing happens delayed and adds energy and synchronization costs.

Polling-Based Messaging. Since a high-throughput message passing between worker threads is crucial for query processing, polling-based (busy-waiting) implementations are preferred to event-driven (blocking) ones to avoid additional scheduling costs and system calls. This leads to an always-on situation where hardware threads never enter a sleep state (C-state).

Elasticity Extensions. To cope with the static mapping and load balancing issue, we propose elasticity extensions for the basic data-oriented architecture. The main concept is to get rid of the static assignment between worker thread and data partitions (cf., Figure 1), which requires significant changes to the message passing layer. Instead of using point-to-point connections between workers, we propose a hierarchical message passing that operates on the intra- and inter-socket level. Within a single socket (intra-socket), messages for the same data partitions are buffered and queued. Worker threads continuously dequeue message batches for a data partition, take ownership of the entire partition; process the messages; and release the partition. The communication between sockets (inter-socket) is handled by a communication thread per socket that buffers messages targeting remote sockets and executes the actual message transfer to the communication thread on the remote socket side. Using this approach, we are able to elastically grow or shrink the number of worker threads and implicitly solve the load balancing issue within a single socket.

ECL Integration. The Energy-Control Loop addresses the polling-based messaging issue (C-states) and furthermore, the appropriate configuration of the core and uncore clocks (P-states). In Figure 1, we visualized the full architecture of our approach including the elastic data-oriented architecture and the ECL. As shown, the ECL is organized hierarchically into one socket-level ECL per physical processor and a single system-level ECL. An integral component of each socket-level ECL is a local energy profile (cf., Section 4), which stores performance and energy efficiency information for a set of hardware configurations for the current workload the respective socket is facing. The socket-level ECL itself is responsible for detecting the current performance demand and for applying the most energy-efficient hardware configuration for this demand as well as for maintaining the energy profile in case of a changing workload (cf., Section 5.1). Since queries are mostly executed on multiple workers on multiple sockets, the system-level ECL is necessary to monitor the current average query latency and to obey a user-defined latency limit. In case of an approaching or already existing latency limit violation, the system-level ECL has multiple measures influence the socket-level ECLs (cf., Section 5.2).

4 ENERGY PROFILES

Based on our findings of Section 2, we abstract the different energy-control features to configurations, which are aggregated to an energy profile. The energy profile is a substantial component of each socket-level ECL, because it describes performance and energy efficiency
4.1 Configurations

A configuration represents a specific system state in terms of hardware energy-control settings for a single processor. Configurations themselves are workload-agnostic, but exhibit different performance and energy characteristics when being evaluated in the context of a specific workload. A configuration comprises:

1. The set of active hardware threads on a processor.
2. The core frequencies of the active physical cores. All other frequencies are set to their respective minimum.
3. The uncore frequency of the processor.

Hence, a configuration is expressed as:

\[ c_x = (\{ \text{HyperThread} \}, \{ \text{core, } f_{\text{core}} \}, f_{\text{uncore}}) \]  

For instance, a configuration \( c_1 \) can be instantiated as:

\[ c_1 = (\{1, 2, 3, 4, 5\}, \{1, 1.2 \text{ GHz}, 2, 2.1 \text{ GHz}\}, 3 \text{ GHz}) \]

This configuration activates the first physical core and both HyperThread siblings as well as the second physical core with one HyperThread. The core frequency of the first physical core is set to 1.2 GHz and the clock of the second core is set to 2.1 GHz. The uncore clock is pinned to 3 GHz. During the evaluation process of a configuration, it is enriched with the following information:

1. The power consumption of the socket, which is measured by the RAPL counters including the Package and DRAM domain.
2. The performance score. Because this score is calculated differently for different workloads, we use the number of instructions retired by all of the active hardware threads on the socket. This performance can be measured with the help of the integrated performance counters of the CPU and is highly correlated to the high-level performance scores we used in our experiments.
3. The energy efficiency, which is calculated as performance score divided by power consumption (W⁻¹).

4.2 Energy Profile Generation

We call a set of configurations the energy profile. The configuration generator is responsible for finding a set of configurations that covers a high variety of distinct hardware configuration states to explore most of the configuration spectrum. In Figure 9, we compared three different parameter settings of the configuration generator. The available parameters are the number of different core frequencies \( f_{\text{core}} \), the number of distinct uncore frequencies \( f_{\text{uncore}} \), the usage of mixed core frequencies \( f_{\text{core-mixed}} \) (off means that all active cores are set to the same frequency), and the maximum number of generated configurations \( c_{\text{max}} \). Using these parameters, the configuration generator calculates all unique configurations taking the homogeneity of the individual cores into account, for instance activating physical core 1 is the same as activating core 2. If the resulting number of configurations is too high, the generator aggregates hardware threads to groups resulting in a decreased granularity of the energy profile.

Energy Profile Granularity. For our experiments we set \( c_{\text{max}} \) to 256 for visualization reasons. Figure 9(a) shows the energy profile for \( f_{\text{core}} = 4 \) (including the lowest, highest, and turbo frequency), \( f_{\text{uncore}} = 3 \), and \( f_{\text{core-mixed}} = \text{off} \) while we run a compute-bound workload (incrementing local counters). The resulting number of configurations is \( |\text{threads}| \cdot |f_{\text{core}}| \cdot |f_{\text{uncore}}| = 288 \). Because \( c_{\text{max}} \) is limited to 256, the generator treats both HyperThread sibling of a physical core as one core group resulting in 144 configurations plus the idle configuration (all cores turned off). The color of the outer circles encodes the average core frequency (ranging from green to red), the inner color the uncore frequency, and the diameter the number of active cores. The x-axis indicates the performance level as the performance score normalized to the peak performance score. The y-axis shows the energy efficiency of a configuration normalized to the measured peak energy efficiency. The profile shows that the lowest frequencies are the most energy-efficient ones for low performance levels until their respective performance potential is exhausted. Moreover, we observe that the lowest uncore frequency is the most energy-efficient one, as it is supported by the experiment in Figure 8. Using the energy profile, the ECL can determine the most energy-efficient configuration for a specific demanded performance level. Thus, only the skyline of the profile is of interest (opaque configurations in the chart). Database systems without energy-control mechanisms usually use all available cores at the highest frequency as long as enough work is available, which is known as race-to-idle (RTI). Therefore, the baseline in the figure shows the respective energy efficiency that is achieved for different performance demands using this approach. Obviously, a more energy-efficient way is using a RTI strategy that switches between...
idle mode and the most energy-efficient configuration, which is depicted as the ECL RTI line. Increasing $f_{\text{core}}$ (Figure 9(b)) to 7 or enabling $f_{\text{core-mixed}}$ (Figure 9(c)) is not significantly improving the skyline of the energy profile, but causes the profile to include more configurations, which are more costly to maintain in case of a workload change. The skyline is not improving, because the original parameter setting already covered the most important supporting points of the configuration space.

Workload Dependency. The purpose of an energy profile is to provide its socket-level ECL with information about performance level and energy efficiency of the configurations. However, especially the performance level of a configuration depends on the current workload of the DBMS. While the compute-bound workload in Figure 9 shows an almost perfect energy profile, real-world energy profiles look much more different. This difference occurs usually as soon as the contention on hardware resources increases. In case of an in-memory DBMS, those points of contention are usually the memory controller or shared cache lines. To demonstrate the effect of memory controller contention, we conducted the energy profile for the memory-bound workload (i.e., a column scan), which is shown in Figure 10(a). This profile looks completely different compared to the one of the compute-bound workload. It shows that high core frequencies are a bad choice, because the bandwidth can not be further increased, and that a high uncore frequency is beneficial in terms of performance level and energy efficiency (the dense clusters in the energy profile). To quantify the impact of cacheline contention caused bottlenecks, we used a workload where all threads atomically increment a single variable. As Figure 10(b) shows, the profile once again looks completely different. Here, the most performing and most energy-efficient configuration uses only two hardware threads at turbo frequency with the lowest uncore frequency. While the maximum possible energy savings amount up to 40% (highest difference between baseline and ECL RTI) for the previous workload, the savings in terms of energy are about 90% in such a scenario with an additional query response advantage of 200%. Nevertheless, since such a workload is very artificial and uncommon for real-world workloads, we show an additional energy profile in Figure 10(c) that was conducted using a workload where multiple threads insert values into a shared hash table. We again observe the same effects at a smaller scale with a potential energy saving of 42% and a query response benefit of about 8% compared to the baseline.

Conclusions. Based on the experiments, we can conclude that the shape of the energy profile can change arbitrarily for different workloads, because contention on hardware resources is the common case in main memory database systems. Moreover, we have shown that choosing the right configuration can significantly improve the energy efficiency, performance, as well as response time.

### 4.3 Ruling Zones

As depicted in Figure 10, the socket-level ECL differentiates three different ruling zones that influence the control strategy:

**Optimal Zone.** This zone hosts only the most energy-efficient configuration and the socket-level ECL is eager to reside in this zone, because the most energy savings are experienced here.

**Under-Utilization Zone.** This zone hosts all configurations left to the most energy-efficient configuration and only energy-efficient skyline configurations are taken into account by the ECL. Since database servers are mostly over-provisioned to cope with load peaks, most of the time is usually spent in this zone where energy efficiency of the configurations is mostly significantly lower compared to the optimal zone [2]. Thus, we apply the ECL RTI method within this zone, which means that the socket-level ECL is frequently switching between idle mode and the most energy-efficient configuration (optimal zone). The potential energy savings for the RTI mode are shown by the ECL RTI line. Using this strategy we are able to partially compensate the high energy costs for activating the first core on a socket (cf., Figure 4). The ECL RTI energy savings are about 40% for very low performance levels and decrease the higher is performance level becomes, because the time the system is able to reside in idle mode is decreasing.

**Over-Utilization Zone.** This zone hosts all configurations right to the most energy-efficient configuration. Because of the decreasing energy efficiency, configurations of this zone are only applied if the optimal zone doesn’t provide enough performance to master the current load within the given query latency limit. The range of the zones depends on the energy profile and thus, on the workload. For instance, for the energy profiles in Figure 10(b) and 10(c), the over-utilization zone is small or not even present.

---

**Figure 10:** Energy profiles using different workloads. $f_{\text{core}} = 4$, $f_{\text{core-mixed}} = \text{off}$, $f_{\text{uncore}} = 3$ (145 configurations). The different ruling zones (under-utilized, optimal, and over-utilized) for the socket-level ECL are highlighted.
5 ENERGY-CONTROL LOOP

In this section, we present the details of our Energy-Control Loop (ECL). The ECL is designed hierarchically as depicted in Figure 1. Each processor independently runs a socket-level ECL maintaining its own energy profile and configuring only the hardware resources available on the respective CPU. Additionally, a system-level ECL is responsible for monitoring the query response times and influences the decision making of the individual socket-level ECLs. In the following, we discuss both components in detail.

5.1 Socket-Level ECL

Since each CPU possesses its own energy counters (RAPL), the lowest available unit for measurements is a single processor (socket). Thus, it is a natural decision to rule the energy tuning features at the socket-level, which is exactly the job of a socket-level ECL. For that reason, there are as many active socket-level ECLs as processors available on the platform, each of them running as a separate thread, that is pinned to the respective processor. Because the workload characteristics can vary per processor, each socket-level ECL uses and maintains its own energy profile to achieve a high accuracy.

Decision Making. The entire ECL and thus, the socket-level ECLs, are designed as a reactive control loop. Hence, the socket-level ECL needs to quickly respond to load changes and is therefore executed periodically at the scale of a second or less (≤ 1 Hz). Figure 11 shows a guiding example that we will leverage throughout this section. The example shows the measured socket utilization and the performance level that is applied by the socket-level ECL over time. In our example, the base socket-level ECL interval is set to one second. The socket-level ECL consists of two controllers, which we will discuss in the following:

Utilization Controller. This controller is responsible for determining the current performance level demand of the DBMS on the respective processor. Hence, the utilization controller continuously receives utilization information of the worker threads from the database runtime. As shown by the example in Figure 11 at the time points 1 s to 4 s, the database runtime reports a full utilization of 100 % forcing the utilization controller to increase the performance level and apply the corresponding most energy-efficient configuration fulfilling the calculated performance demand based on the local energy profile. Since the utilization can only be measured relative to the amount of active worker threads, the utilization controller is not able to exactly determine the required performance level in case of a full utilization. Thus, the utilization controller uses a discovery strategy that exponentially increases the performance level in each socket-level ECL call to avoid activating more hardware resources than necessary on the one hand and to cope with sudden load spikes on the other hand. Additionally, the discovery strategy considers information reported from the system-level ECL. The opposite scenario is a utilization below 100 % as it happens for instance at the time points 5 s and 6 s in the guiding example. In this case, the utilization controller is able to determine the required performance level using this formula:

\[
\text{performance level}_{\text{new}} = \text{utilization} \cdot \text{performance level}_{\text{old}} \tag{3}
\]

Once again, the system-level ECL uses the calculated performance level and applies the most energy-efficient configuration satisfying the performance level known by the energy profile.

Race-To-Idle (RTI) Controller. The RTI controller leverages the information reported by the utilization controller (i.e., new performance level and configuration) and decides whether to use a race-to-idle strategy or not. There are two reasons for applying RTI: (1) to partially compensate the high costs for activating the first core on a socket and (2) to emulate any performance level for which no configuration is known by the energy profile. In practice, the RTI controller always tries to switch between the most energy-efficient configuration and idle mode when the performance demand is in the under-utilization zone. However, the negative side effect of applying RTI is that the query response time is negatively affected if the system resides for a long time in idle mode. Thus, the RTI controller switches at a high frequency (e.g., every 10 ms) or disables RTI at all. Additionally, the RTI controllers of different socket-level ECLs try to synchronize idle times, because a socket can only enter its deepest sleep state, if all sockets of the machine are in idle mode (cf., Figure 4). In the example, we schematically demonstrate the RTI usage between time points 7 s and 9 s. At time point 7 s, the configuration of the previous interval is emulated using RTI switching between the most energy-efficient configuration available (optimal zone) and idle mode. At time point 8 s, the utilization controller detects a lower utilization and reduces the performance level accordingly. Hence, the RTI controller causes the system to spend more time in idle mode and uses three instead of two RTI cycles per socket-level ECL interval to keep the response time low. In practice, RTI controller does up to 50 RTI cycles per 1 s socket-level ECL interval.

Energy Profile Maintenance. As demonstrated in Section 4, the shape and skyline of the energy profile depends on the current workload the DBMS respectively processor is facing. Thus, the socket-level ECL needs to quickly adapt the energy profile in case
of a changing workload. If the energy profile in not accurate, the socket-level ECL is not able to calculate the current performance level, RTI calculations become inaccurate, and possibly energy-inefficient configurations are applied. For that reason, it is important how fast configurations can be reevaluated at runtime. Since the speed of applying new configurations and measuring the corresponding energy and performance counters is hardware dependent, the ECL does a meta calibration step once on startup. In this meta calibration step, the ECL detects the times needed for applying a configuration and for measuring the counters. The ECL starts by taking a reference measurement using a generous amount of time and is decreasing the times step by step, while measuring the deviation from the reference measurement. This process happens first for the measure time followed by the apply time. During the measurement, the ECL switches between the highest configuration (all cores at highest frequency) and the lowest available configuration (one core at lowest frequency). As presented in Figure 12, for applying a configuration, the evaluation is even accurate when using a 1 ms interval. Recent works [7] support this finding and measured an overhead of some µs for C- or P-state transitions. However, the time for measuring the counters becomes more and more inaccurate when being decreased. The source of most of the deviation we encountered, was the RAPL measurement, when switching to the lowest configuration. We identified a measurement interval of 100 ms to be the best trade-off between accuracy and speed. To adapt the energy profile at runtime, we use two strategies:

**Online Adaptation.** This strategy is continuously used to adapt to workload changes. Every time the socket-level ECL applies a certain configuration, it measures the power and performance metrics using the respective performance counters (i.e., RAPL and instructions retired) and updates the configuration in the energy profile. The main advantage is that no overhead is generated and currently used configurations are highly accurate. However, the obvious drawback is that only configurations are maintained which are reported by the energy profile to be the most energy-efficient ones, which may not be the case anymore.

**Multiplexed Adaptation.** This strategy complements the online adaptation and is triggered as soon as a high drift in configuration accuracy is detected and reevaluates all stale configurations of the energy profile. To only minimally affect the operation of the system, the multiplexed adaptation once again leverages the capabilities of the RTI controller to simulate high load situations, which can enlarge the time fraction that is used for adaptation within one socket-level ECL period (between time point 11 s and 12 s in the example). As discussed, both energy profile adaptation strategies work hand in hand and exhibit a different behavior in terms of overhead, invasiveness, and quality of the resulting energy profile. As we will show in our end-to-end evaluation, even the multiplexed adaptation only minimally affects the operation of the ECL and the combination of both strategies successfully maintains the energy profile.

### 5.2 System-Level ECL

While socket-level ECLs are responsible for managing a single socket of the system, the system-level ECL manages metrics that are only globally available. In our scenario, this metric is the average query latency, which is the result of the performance of all available sockets in the system. Thus, all socket-level ECLs implicitly influence this metric. The overall goal of the system-level ECL is not to keep the response time as low as possible, instead it takes a user-defined maximum response time, which is considered as a soft constraint, because a reactive control loop is not able to guarantee that this limit is not violated and thus, uses a best effort strategy. The system-level ECL continuously monitors the actual average query latency and calculates its current trend. Based on the trend, it is able to estimate the time until the latency limit is violated. This time is provided for all socket-level ECLs, which use this value to (1) adjust the aggressiveness to enter a higher performing configuration in case of a full utilization and (2) to adjust the RTI usage and switching frequency, since RTI negatively affects the latency. However, a low value of this time (or even zero if the limit is already violated), does not mean that socket-level ECLs automatically ramp up all available hardware resources, because the work can be unevenly distributed across the sockets. Thus, socket-level ECLs still select lower performing configurations if a lower utilization is reported, but are more eager to increase the performance level on the respective socket.

### 5.3 Architectural Dependencies

Within this paper, we primarily focus on a data-oriented architecture, because of its superior scalability originating from the local and latch-free data structure access at partition level. This scalability property is a major prerequisite for energy efficiency and energy proportionality. Nevertheless, the basic concept of the ECL is applicable to transaction-oriented database systems, but requires additional research to address critical details such as (1) spinlocks, which – in contrast to data-oriented systems – often occur and tamper with our performance metric (instructions retired) and (2) cross-socket interferences leading to highly frequent energy profile adaptations. Since energy savings are mostly a result of hardware or software scalability limitations as they are common for such architectures, we believe this is a promising research direction, too.
6 END-TO-END EVALUATION

In this section, we evaluate (1) the ability of our ECL to adapt to changing database loads while obeying a latency constraint, (2) the overall energy savings for a multitude of load profiles and workloads, and (3) our energy profile adaptation mechanisms in case of a workload change. All experiments were conducted on the 2-socket Xeon E5-2690 v3 presented in Section 2 running a data-oriented in-memory DBMS (anonymized), which implements our proposed architecture including the ECL (cf., Section 3). The worker-partition ratio is set to 1:1 for all experiments.

Workloads and Load Profiles. Throughout our evaluation, each experiment uses a combination of a workload and load profile. The workload specifies the queries as it is done by standard database benchmarks. Additionally, we use load profiles that define the number of queries per second sent to the database system over time, because energy efficiency depends on the load. Table 1 enumerates all workloads and load profiles we considered for our evaluation. As workload, we employ the TATP (OLTP) [9], SSB (OLAP) [17], and a custom key-value store benchmark each either fully indexed or not indexed at all. The key-value store uses 4 Byte keys (uniformly distributed) and values and is either memory latency-bound in the indexed case or memory bandwidth-bound in the non-indexed case and thus simulates typical access patterns that have a completely different energy profile. For instance, the energy profile of the non-indexed version resembles the one in Figure 10(a). As load profile, we employ the spike profile, which covers the full range of load situations, and the twitter [1] load profile, to investigate how the ECL acts in real-world database load situations.

6.1 Load Adaptation and Query Latency

In this section, we investigate the behavior of the ECL in case of a changing database load and focus on the power draw as well as the query latencies over time. For all experiments, we use the non-indexed key-value store workload and different load profiles. The query latency limit is set to 100 ms and the ECL base frequency is 1 Hz (1000 ms between calls) respectively 2 Hz (500 ms). For power measurements we leverage the integrated energy counters (RAPL) and the baseline uses all available hardware threads with CPU and OS frequency control resembling in a race-to-idle strategy.

Spike Load Profile. Figure 13 shows the results for the spike load profile, which was run for 3 minutes. The respective database load over time (load profile) as well as the power measurements for baseline and ECL are visualized in Figure 13(a). The first observation is that the ECL never draws more power than the baseline, because only the most energy-efficient configurations are applied. Our next finding is that the ECL significantly improves energy proportionality, especially in load situations above 50 % (20 kQps) where it is almost perfect. In lower load situations (<50 %), energy proportionality gets worse because of the static power consumption of the processors, which gets less significant at higher loads. Starting at 80 s, the spike load profile generates an overload situation meaning that the system receives more queries than it is actually able to handle. An interesting observation is that the baseline stays for about 50 s in the overload state, while the ECL only resides for about 20 s there and still draws less power. The reason is that using all available hardware resources (baseline) provides less performance compared to the performance of the configuration that is selected.
by the ECL, because more contention on the memory controllers is generated for this specific workload. As shown by Figure 13(b), the ECL stood within the response time limit of 100 ms and the response time limit violations happened only within the overload situation. Moreover, we observe that an increased ECL base frequency (2 Hz) only slightly improves the query latencies.

**Twitter Load Profile.** Figure 14 shows the results for the tweet load profile. This profile includes sudden load peaks and is frequently alternating between increasing and decreasing the system load. The load and power measurements are shown in Figure 14(a). Once again, we observe that the ECL draws significantly less power compared to the baseline most of the time. However, we also observe that the ECL takes more time to adapt the hardware configuration to the sudden load peaks, because of its reactive nature. This finding is supported by the latency measurements (1 Hz) depicted in Figure 14(b). Here, we observe outliers caused by sudden load peaks, which can be reduced by increasing the ECL base frequency to 2 Hz. However, it’s worthwhile to mention that we replayed a 2 hours load profile within 3 minutes.

### 6.2 Energy Savings

Our main objective is to increase the energy efficiency of the DBMS meaning a decreased overall energy consumption. Hence, we measured the total energy consumption of all workload and load profile combinations for the baseline as well as the ECL and calculated the energy savings. Since we used energy counters for our measurements, we present only the relative energy savings, which accurately correlate to the relative power supply unit (PSU) measurements [7]. Table 1 lists the relative energy savings and the most energy-efficient configuration for each workload (average for SSB), which is mostly static for the same workload and is a good hint to reason about the results. The full energy profiles for TATP and SSB workloads are placed in the appendix. As shown, we experienced the most energy savings for non-indexed workloads, because the memory controllers become a bottleneck due to parallel column scans that saturate the memory bandwidth. In this case, our custom key-value store benchmark achieves the most energy savings, because it solely executes column scans and thus requires only a low hardware thread count at the lowest frequency. In contrast, TATP and SSB need to communicate with other partitions (i.e., joins) and reconstruct tuples, which favors more hardware threads at a medium frequency and thus reduces the potential energy savings. The measured energy savings for indexed workloads reach from 15.8 % to 23.4 % originating from medium core frequencies and a generally lower uncore frequency as a result of the memory latency-bound access pattern. Nevertheless, we also observe that the SSB benchmark requires in average a higher uncore clock, because of the increased data volume that needs to be shipped between partitions (depends on the query). Moreover, we observed that the ECL itself only consumes 2 % of the compute time of a single hardware thread per socket, which is a negligible number.

### 6.3 Energy Profile Adaptation

While the previous experiments assume a static workload, we now focus on energy profile adaptations caused by changing workloads. In our setup, we suddenly switch from the indexed key-value store benchmark to the non-indexed one, which resembles a major workload change. The database load is fixed to 50 % with 1 Hz ECL base frequency. We use three ECL settings to demonstrate the effect of the different energy profile maintenance strategies (cf., Section 5.1): no adaptation (ECL static), online adaptation (ECL online), and multiplexed adaptation (ECL multiplexed; includes online adaptation).

Figure 15 visualizes the power consumption over time and the total energy draw for the different adaptation strategies. The workload switch happens at 40 s. When looking at the ECL static measurements, we already observe a slightly increased power consumption before the workload switch, because online adaptation is disabled and the profile is not adjusted to small variations that usually occur even for a static workload and mostly originate from the hardware...
itself. Immediately after the workload switch, the power consumption is higher compared to the other maintenance strategies and that the numbers are fluctuating, because the socket-level ECLs are not able to (1) accurately determine the performance level, to (2) find the appropriate configurations, and to (3) do accurate RTI calculations. The ECL online measurements show that online adaptation quickly adapts the energy profile allowing the socket-level ECLs to do accurate configuration decisions. However, since the online adaptation only reevaluates configurations that were applied, it is the save way to reevaluate stale configurations of the energy profile, which is done by the multiplexed adaptation. As the ECL multiplex measurements show, this process requires more time, but manages to find a slightly more energy-efficient configuration for the new workload. As assumed, the ECL static setting without any energy profile adaptation draws significantly more energy and is mostly not able to stay within the query response time limit as shown by Figure 16. In contrast, the ECL online and ECL multiplex settings consume about 25% less power and are able to stay within the response time limit. Thus, we can conclude that the active energy profile maintenance at runtime is critical for a static workload and especially for changing workloads.

7 RELATED WORK
The energy efficiency of database servers is a critical research topic [8, 19], because the scalability of database servers is limited by the “energy wall”. We focus on the energy analysis of database systems and on the active usage of hardware energy-control knobs.

Energy Analysis. Early works started with analyzing the potential of DVFS to increase the energy efficiency of a database server [26]. The main conclusion was that this approach is not feasible, because of the high static power consumption of the hardware that was available at that time and thus the authors concluded that the most energy-efficient configuration is the most performing and power consuming one and that energy optimizations are only feasible at cluster level. As we showed in Section 2, these findings do not hold anymore for current servers. Other studies [5, 28] found that software optimizations (i.e., the query optimizer) can significantly improve energy efficiency of a DBMS by considering energy as an additional optimization goal. While the previous works did their analysis at query-level, Psouridakis et al. [22] analyzed the energy consumption of individual database operators and concluded that fine-grained scheduling mechanisms are able to further improve the DBMS energy efficiency. However, all of these works did their energy analysis either on outdated hardware or did not consider the additional energy-control knob offered by a current system such as independent core and uncore frequencies as well as the EPR.

Active Energy-Control. In the context of distributed database systems, several approaches [15, 16, 24] tried to achieve energy proportionality by dynamically powering individual servers down or up. Because of the high costs for moving data and power cycling single servers, those approaches are only applicable as long term solutions and negatively affect energy efficiency, since data movement consumes a high amount of energy and scale-up architectures usually exhibit a better performance compared to scale-out solutions. Within a single database server, some works [14, 28, 29] dealt with adding energy as an additional optimization criteria to the query optimizer of the MySQL or Postgres DBMS. To do so, the authors built and calibrated energy and performance models for single page operators that are leveraged by the optimizer. The energy savings amounted to about 20 %, but did not consider a response time limit and the evaluation was limited to a dual-core system, since MySQL and Postgres are disk-based DBMS with limited intra-query parallelism. In this context, energy response time profiles (ERP) [14] were proposed to quantify the trade-off between query performance and energy consumption sharing some analogies with our proposed energy profiles. However, our energy profiles (1) operate close to the hardware instead on the level of operators, (2) have a high resolution in terms of possible configurations, (3) are hardware and DBMS implementation independent, (4) consider mutual interferences of simultaneously running queries, (5) do not require any hand-crafted energy and performance models, and (6) automatically maintain themselves. Another class of active energy-control approaches uses a feedback control loop to dynamically adjust DVFS settings at runtime. For instance, Tu et al. presented the PostgreSQL extension E2DBMS that adaptively controls the DVFS setting (one per processor) and the power state of the hard disks, while obeying a query throughput target. The authors assume a monotonic relationship between power and performance, which is mostly not the case as we demonstrated in Section 4. Another approach [30] relies on a workload classifier that uses the amount of disk I/O to select the appropriate DVFS setting, since a high disk I/O rate makes a lower CPU power mode more energy-efficient. The recent LAPS [13] approach employs a feed-forward control mechanism that tries to stay within a certain response time limit. LAPS operates at the core level and adjusts the DVFS setting per core and is able to cope with short-term workload fluctuations. Additionally, the authors propose memory sizing as an additional tool for energy control. However, the main drawback of LAPS is that a precise query execution time model is needed, which is hardware and DBMS dependent and should consider other queries that are running in parallel. Moreover, this approach is not suitable for intra-query parallelism. Compared to our ECL, the mentioned works mainly address fairly parallel disk-based transaction-oriented database systems where disk accesses are the main bottleneck. Contrary, our ECL primarily focuses on data-oriented in-memory DBMS that run on massively parallel scale-up hardware and bottlenecks usually occur within processors and memory controllers.

8 CONCLUSIONS
Energy is the key- limiter for the scalability of scale-up database systems. This paper aimed at reducing the energy consumption of data-oriented in-memory database systems that make heavy use of the main power consumers, namely CPU cores and main memory. Thus, we proposed the ECL as a holistic software-based approach for adaptive energy-control that obeys a query latency limit as a soft constraint and actively optimizes energy efficiency and performance of the DBMS using adaptive energy profiles. In our evaluation, we observed energy savings of up to about 40%.

9 ACKNOWLEDGMENTS
This work is partly funded within the DFG-CRC 912 (HAEC).
REFERENCES


A BENCHMARK ENERGY PROFILES

In this section of the appendix, we present the energy profiles that correspond to the measurements given in Table 1 of Section 6. In particular, we present the energy profiles for the TATP and the SSB benchmark each in the fully indexed and the non-indexed version. In case of the SSB benchmark, we show the energy profile of query 2.1 as representative.

Figure 17: Energy profile for indexed TATP.

Figure 18: Energy profile for non-indexed TATP.

Figure 19: Energy profile for indexed SSB (Q2.1).
Figure 20: Energy profile for non-indexed SSB (Q2.1).

The general observation we derive from the energy profiles is that the indexed version of the TATP (Figure 17) and SSB benchmark (Figure 19) look similar in terms of their general shape. We also observe the same similarity when comparing the non-indexed version of the TATP (Figure 18) and SSB (Figure 20) benchmark. Those similarities originate from the join-intensive workload pattern that is shared by both benchmarks. However, a closer look reveals that there are small differences between both benchmarks.

Another general observation is that the general shape of the indexed benchmark versions is very related to the compute-intensive energy profile visualized in Figure 9(a), which causes almost no hardware bottlenecks. In contrast, the indexed benchmark versions generate a low contention on the memory controller, which is represented by their energy profiles and causes the difference to the compute-intensive energy profile. The same analogy is obvious when comparing the energy profiles of the non-indexed benchmark versions to the memory-intensive energy profile shown in Figure 10(a). Here, the same configuration cluster of low uncore frequency configurations is shared. However, for medium and high uncore frequencies, this configuration cluster is wider spread compared to the energy profile of the memory-intensive workload.