Datenbankspektrum manuscript No.
(will be inserted by the editor)

WattDB - A Journey towards Energy Efficiency

Daniel Schall · Theo Härder

Received: date / Accepted: date

Abstract Due to their narrow power spectrum between idle and full utilization [2], satisfactory energy efficiency of servers can only be reached in the peak-performance range, whereas energy efficiency obtained for lower activity levels is far from being optimal. Hence, this hardware property obviates a desired energy proportionality or minimal energy use for the entire range of system utilization. To approximate energy proportionality for all activity levels, we developed various versions of WattDB, a distributed DBMS, which runs on a dynamic cluster of wimpy computing nodes. In this survey, we sketch important design decisions and implementation steps towards the final state of WattDB. For these reasons, we discuss our findings on a cluster with dedicated storage nodes and static data allocation, on dynamic data repartitioning and allocation, and on a dynamic cluster where each node can serve as storage and processing node in a symmetric way. Our experiments show that WattDB dynamically adjusts to the workload present and reconfigures itself to satisfy performance demands while keeping its energy consumption at a minimum. Finally, we compare the performance and energy results of the WattDB software running on the cluster of wimpy nodes with that of a brawny server.

Keywords Energy efficiency · Energy proportionality · Cluster of wimpy computing nodes · Distributed database management · Data partitioning

D. Schall · T. Härder
University of Kaiserslautern
P.O. Box 3049, 67653 Kaiserslautern, Germany
E-mail: {schall,haerder}@cs.uni-kl.de

1 Introduction

The need for more energy efficiency in all areas of IT is not debatable. Besides reducing the energy consumption of servers, other ideas like improving the cooling infrastructure and lowering its power consumption help reducing the energy footprint of data centers. In the area of DB management, initial research towards energy efficiency primarily focused on the use of flash memory or solid-state disks (SSDs) as a disruptive storage technology. However, solely replacing conventional disks (HDDs), provided only limited gain, because the main energy consumers of a computing system—main memory and CPU—were hardly considered in such energy saving approaches. On the other hand, Tsirogiannis et. al. [19] observed in an extended study using empirical experiments that *the most energy-efficient configuration of a single-server DBMS is the highest performing one.* But this is only true, if we exclusively focus on performance benchmarks or on specific applications where the DBMS steadily runs in the peak-performance range. Typical DBMS activity levels, or their average system utilization, are much lower and their peak utilization may be only touched for some minutes or at most a few hours per day. For those computing demands, energy use of a single-server DBMS is suboptimal and far from being energy proportional.

Due to their narrow power spectrum between idle and full utilization of single-node servers [2], the goal of satisfactory energy efficiency cannot be reached using today's (server) hardware. Reducing energy usage of servers to a sufficient level leads to a demand for energy-proportional hardware. Because such a goal seems impractical, we should at least aim at an emulation of the appropriate outcome at the system level.

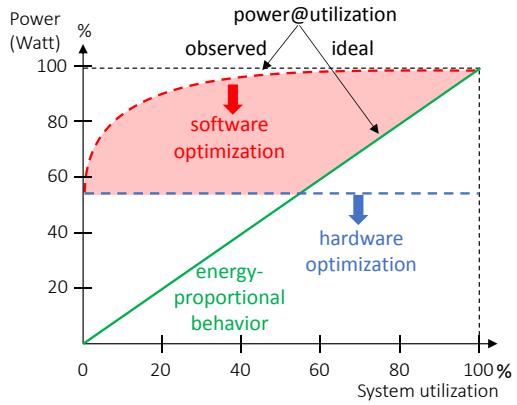


Fig. 1 Power use over single-node system utilization

In contrast to centralized, brawny servers, a scale-out cluster of lightweight (wimpy) servers is able to shutdown single nodes independently. At an abstract level, this enables the cluster to dynamically add storage and processing power based on the cluster's utilization. Similar to cloud-based solutions, we hypothesized that a cluster of nodes can adjust the number of active nodes to the current demand and, thus, approximate energy proportionality. Based on these observations, we developed WattDB running on lightweight, Amdahl-balanced nodes using commodity hardware. The cluster can dynamically shrink and expand its size, dependent on the workload. Reconfiguring a cluster to dynamically match the workload requires data to be redistributed among the active nodes to balance the utilization. Yet, copying data is time-consuming and adds overhead to the already loaded cluster. Therefore, reducing both, time and overhead, was a crucial objective to achieve an elastic DBMS. Although the cluster is not be as powerful as a monolithic server, our system consumes significantly less energy for typical workloads.

We will use the term *power* (or power consumption) to denote the current consumption of electricity; hence, the unit of power is *Watt*. The power consumption of a server/component over time is called *energy* (or energy consumption), which is expressed in *Joule* (*Watt * sec*). In the DB community, and for the TPC-* benchmarks as well, the number of transactions – defined as specific units of work addressed by the TPC-* workloads – has prevailed as an application-related measure for the quantity of computations: *transactions per second (tps)*. To express how efficiently a certain set of queries can be processed using a given amount of energy, we use the term *energy efficiency (EE)*:

$$EE = \frac{\#\ of\ transactions}{energy\ consumption} = \frac{tps}{Watt}$$

Recently, power consumption of (DB) servers and energy efficiency became important factors too. A study revealed, that the average server consumes about as much power to draw energy cost to level with its acquisition cost in a five year period [11]. Therefore, energy-efficient solutions are drawing more and more attention. Barroso et. al. examined the power profile of typical servers at Google in [2] and concluded, that the typical server is mostly running between 10% and 50 % of its peak utilization, often idle and hardly runs at 100% load. Other studies suggest similar usage patterns [15].

In Figure 1, the power consumption of a server is plotted against its utilization from idle to highest. Here, utilization refers to the system load, i. e., 100% utilization correspond to a system fully utilized, at maximum CPU and disk use. While the server needs the most power at high utilization, it already consumes ~50% of its peak power. With increasing workloads, power consumption quickly converges to its peak. Even low workloads require a disproportional high amount of power.

The findings of Barroso et. al. are representative for all of today's server systems, with small variations based on individual configurations. Because today's hardware components exhibit similar power/performance figures, building a more energy-efficient, monolithic system seems impossible. In summary, (DB) servers are typically working on moderately sized workloads, far from peak utilization, but require an unproportional share of power for such workloads. Therefore, research began looking for optimizations of the observed effects and focus shifted from a purely performance-centric view to include energy-related aspects as well.

First, existing solutions were optimized to adjust to workloads and to better utilize the given hardware infrastructure. In [1], Albers studied energy savings by explicitly powering down components. Similarly, Wang et. al. examined techniques for reducing power consumption of stand-alone servers, like DFVS (Dyn. Frequency and Voltage Scaling) in modern processors for their saving potential in [20]. Although performance can be traded for power consumption, no "big leaps" could be identified [10]. Tsirogiannis et. al. [19] observed in an empirical study that *within a single node intended for use in scale-out (shared-nothing) architectures, the most energy-efficient configuration is typically the highest performing one.*

Based on previous observations, Lang et. al. [9] ran experiments on a cluster of lightweight nodes, concluding that the cluster's size can adjust to performance needs (with "friction losses") and that it is indeed possible to achieve better energy efficiency by taking a clustered approach. Other approaches, featuring DB systems running on the cloud, were also promising lower energy consumption by dynamically scaling to more or fewer nodes. Yet, when our research began, no publicly

available, distributed DBMS was able to run on a dynamically changing number of nodes and autonomously adjusting itself to a given workload.

In this paper, Sect. 2 explains our measurement equipment and sketches a new energy-centric benchmarking paradigm, which we have proposed to extend TPC-Energy. In Sect. 3, we have summarized design considerations and implementation aspects of WattDB, before we outline the historic development steps of WattDB. Sect. 4 concentrates on the storage layer and gives an overview of I/O-related experiments using simple read/write requests. We focus on the query layer (Sect. 5), where we explore the performance impact of distributing operators among cluster nodes. Using the lessons learnt so far, we extended WattDB to handle a truly dynamic cluster where data and queries were dynamically allocated (Sect. 6) Since data repartitioning turned out to be costly, we explored in Sect. 7 the performance effects of partitioning schemes. Finally, Sect. 8 compares our cluster with a big server w. r. t. performance and energy consumption, before Sect. 9 concludes our work and finishes WattDB's journey towards energy efficiency.

2 Quantifying Energy Consumption

Traditionally, standard server hardware does not provide integrated probes to measure energy use. Recently, Intel updated the specifications of their Performance Counter Monitors (PCM) to include energy-related measures for CPU and memory as well [21]. Yet, other components are not monitored. Hence, an external solution is needed to accurately measure power use of the system under test (SUT). We have developed a framework for a single DB server, able to assess power consumption per component, and to measure the power consumption of each DB server in a cluster of nodes.

2.1 Measuring Equipment

The measurement framework consists of three key components: First, *probes* are inserted into the electric cables of the device, to measure current and voltage. The measured data is converted by an *A/D converter* to digital signals. The signals are then sent via USB to a connected *PC*, logging and displaying the readings. This PC is also running a benchmark to drive workloads on the cluster and is able to correlate power figures with performance results from the benchmark. This setup is able to meter both performance and energy consumption of the SUT running a multitude of workloads.

The first device for measuring the power use of a single server intercepts the power rails of an ATX power supply to measure voltage and current. We are able to track the power draw of the mainboard, including CPU and main memory, and attached HDDs/SSDs. By using current transducers for measuring, the device can deliver measurements with 1% accuracy. More importantly, the measurements do not interfere with the running SUT. With current transducers, we can also measure alternating current, i. e., the power consumption of a node cluster. For this, a second measurement device was built, able to track up to 10 nodes with maximally 250 Watts each. This device is also monitoring voltage and apparent power of the setup. By replacing the first measurement device with the second, we can re-use the A/D converter and connected PC to now log and analyze power consumption and performance of a cluster.

2.2 A New Benchmarking Paradigm

Traditional DB benchmarks, e. g., TPC-C and TPC-H, are purely performance centric. In addition, the pricing specifications denote acquisition and support costs for the systems under test. With the emerge of TPC-Energy, requiring to report idle power and energy use under full load, energy-related aspects arrived in benchmarking. Yet, TPC-Energy is only an addition to performance benchmarks and does not report power consumption for realistic workloads. So far, benchmarks only determine peak performance. Using TPC-Energy, peak power consumption is also measurable. Yet, as illustrated earlier, realistic workloads do not steadily use the full hardware potential. Typically, peak performance is only required for short periods of time.

Since power consumption does not scale linearly with the workload, energy consumption for a typical workload cannot be deduced from idle and peak consumptions. Therefore, more detailed measurements are necessary. In [15], we developed a benchmarking paradigm to cover the full range of a given DB server and measure its energy consumption at all workloads. We proposed measurements running static workloads (at fractions of the server's peak capacity) to assess each utilization level with its power consumption. Further, we suggested dynamically changing patterns to identify the system's ability to react to changing loads. With this approach, energy efficiency can be quantified and compared.

3 WattDB

Based on the insight, that a single server will not become energy-proportional, we started working on our idea of a distributed DBMS running on a cluster of lightweight nodes. At the time the project started, no

state-of-the-art DBMS was able to dynamically adjust to a workload by scaling-out and back in, respectively.

3.1 Architectural Design Considerations

Our fundamental design requirements—primarily dictated by the goal to save energy as much as possible, but still providing acceptable performance—can be summarized as follows:

- The distributed DBMS should run on a cluster of wimpy nodes for which commodity hardware should be used, i. e., we wanted to compose our cluster with out-of-the-shelf, low-energy components.
- All nodes (except a dedicated one) should be able to enter or leave the cluster at any time.
- All DB data must be accessible from every active node at any time.
- High degrees of parallelism among read/write transactions should be facilitated, where ACID quality has to be guaranteed.
- Parallel and distributed work for single transactions should be enabled, where load balancing among the active nodes in the cluster should be achieved.

Given these objectives, we briefly want to figure out the base architecture for our cluster using the conventional classification of distributed DBMS architectures well-known since the 1980s. *Shared Everything* can be immediately excluded as a conceivable solution—mainly because of the required cluster elasticity, the specialized hardware needed, and the lesser stability/resiliency in case of failures [7]. The operating principle of *Shared Disk* is to fetch all data referenced directly to the node where the transaction code is handled just like a local execution within a centralized DBMS, thereby prohibiting parallelism and distribution of the transactions's work. But such an architecture would perfectly satisfy the data accessibility demands, greatly assist all scalability aspects of computing power and data volume, and facilitate failure recovery. However, direct connectivity of the *shared disks* to a growing number of computing nodes would become very complex, require specialized connectivity hardware, and force DB buffer coherence control among the computing nodes.

Shared Nothing relies on distributed execution of all transactions, because (in its purest form) data is processed at the node where it is stored, i. e., *the load follows the data*. All cooperation/communication tasks among the cluster nodes are handled by messages, which makes scalability of computing power very simple. In turn, scalability of data volume is not for free. But data allocation has to follow the cluster dynamics anyway,

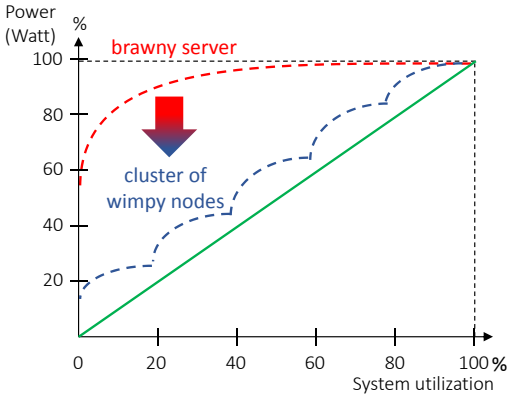


Fig. 2 Power use over system utilization

such that sophisticated data partitioning and movement facilities are necessary in any case. For these reasons, the base principle of WattDB is *Shared Nothing* [7]. Nevertheless, the entire database must be accessible by all active computing nodes. As a consequence, we need to emulate an I/O architecture, where—at each point in time and each cluster configuration—all DB data stored on external devices (SSDs or HDDs) can be dynamically shared by all active nodes, i. e., the shared-nothing processing architecture of the cluster has to be supported by some properties of a shared-disk I/O architecture. This is mainly achieved by dynamic repartitioning/data movement among the external storage devices and transferring record sets among the nodes—thereby acting as *record servers*.

3.2 Architecture Overview

A cluster of lightweight nodes, capable of dynamically powering up and down to scale to the required size, promises better energy efficiency by approximating energy-proportional processing. Fig. 2 tries to explain how a cluster of five nodes could replace a brawny server thereby better approximating energy-proportional processing behavior. Assume the monolithic server runs at a utilization level of 20% (40%) and consumes 85% (95%) of its maximal power. If one (two) cluster node(s) running in peak could take over this load, we could reduce the power consumption to process this load to 25% (45%) and, in turn, emulate better energy proportionality of a brawny server. Of course, the relationships shown in Fig. 2 are idealized, because power consumption of the big server and the full cluster may differ to a certain amount. Furthermore, transaction throughput and runtime achieved by the dynamic cluster may no perfectly match the server's performance at the various utilization levels. Nevertheless, a cluster, automatically ad-

justing its size to the workload, may come closer to the ideal of energy proportionality (see also Fig. 12(b)).

To verify our hypothesis and quantify conceivable differences of performance and power/energy consumption, we use an experimental cluster, consisting of 10 identical nodes. Each node in the cluster is equipped with 2GB of DRAM, an Intel Atom D510 CPU and up to four storage devices. By choosing lightweight, commodity hardware, Gigabit-Ethernet is sufficient to connect the nodes and consider the whole cluster Amdahl-balanced, i. e., its CPU power, memory capacity, and I/O bandwidth reasonably match. With more powerful hardware, faster interconnects and more storage disks would be necessary, driving up cost without giving additional insights. In theory, the same approaches verified on a lightweight cluster can be applied to more heavy-weight nodes.

Each of the nodes in the cluster consumes about 20 Watts when idle, another 3 Watts per storage disk (depending on the model) and up to 35 Watts with all four disks and CPU at full utilization. When powered down in standby, a node only consumes about 2.5 Watts.[1] The connecting Ethernet switch, adding another 20 Watts, is included in all measurements.

WattDB[2] is a newly developed DBMS research prototype, written in C/C++. It accepts SQL queries, but has only a limited set of supported features, e. g., selection, projections, and joins are implemented, as well as sorting, group-by and aggregation operators. Not supported are, for example, sub-queries, multi-statement transactions and conditional expressions. This limitation is not architecture-intrinsic, it is merely by choice, since we did not need to implement fully SQL99-compliant code to verify our ideas.

In WattDB, a dedicated node ("master" node) acts as coordinator for the whole cluster. This is the only node accepting client connections. The master node keeps information about all other nodes, e. g., table partitioning and performance data. Because of its holistic knowledge, this node is performing query optimization and triggers redistribution in the cluster to scale out (or in, respectively). All nodes (including the master) can process queries and store DB data.

To support elastic scale-out and -in, all nodes in the cluster are able to access the entire storage space (by requesting remote pages) and may participate in query processing. Figure 3 depicts a high-level view on a cluster of three nodes, sharing disk space and query evaluation. By dynamically adding and removing (unused) nodes from the cluster, storage space and IOPS, as well as memory capacity and query processing performance can be tuned. To support dynamic reorganization, all tables in WattDB are subdivided into partitions as illustrated in Fig. 4. **Tables** are purely logical constructs, keeping meta-data definitions in place and are stored on the master node. Each table is composed of k *horizontal partitions*, each belonging to a specific node, responsible for query evaluation, data integrity (logging) and access synchronization (locking). The partitioning scheme used is application-dependent, as some applications may benefit from distinct key ranges, while others may prefer scattered data. **Partitions** contain 1 to m *segments*, which are physical units of storage. Each segment is located on a specific disk on a node in the cluster. Partitions are by default index-organized w. r. t. the primary key and support additional, secondary indexes. **Indexes** are realized using B*-trees and span only one partition at a time. Hence, indexes are stored on the same partition as the data and do not contain cross-references to other partitions. **Segments** consist of 4096 blocks or pages, which are consecutively stored on disk. They have a fixed size of 32 MB and are the unit of distribution in the storage subsystem. Hence, all pages belonging to the same segment are copied/moved to other nodes in one batch.

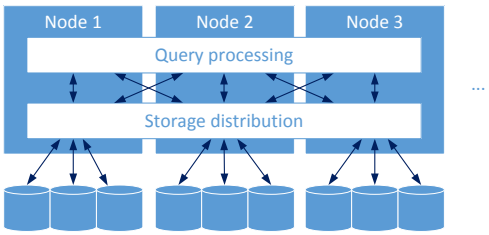


Fig. 3 Conceptual view of the cluster

Cost of reorganization Moving data is an expensive task, in terms of energy consumption and performance impact on concurrently running queries. Data reorganization binds some computing resources, which would be needed to optimally process the query workload. This resource contention leads to fewer resources for the workload and, in turn, reduces query throughput. However, the reorganization cost should amortize by reducing the energy consumption of subsequent queries. Though it is difficult to calculate the exact energy consumption of a data move operation w. r. t. the impact of running queries, the energy cost can be estimated by the duration of the move operation and the (additional) power consumption. Hence, moving 1 GByte of data to

[1] It is also possible to turn nodes completely off, resulting in about 0.5 Watt of power use. But then, start-up times are much higher than just waking up from suspend-to-RAM.

[2] The initial version of the core engine was developed for the 2010 SIGMOD Programming Contest, where it won the second prize [5].

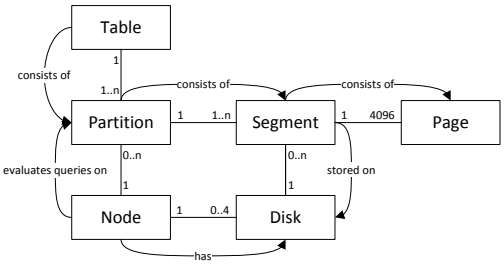


Fig. 4 Database schema

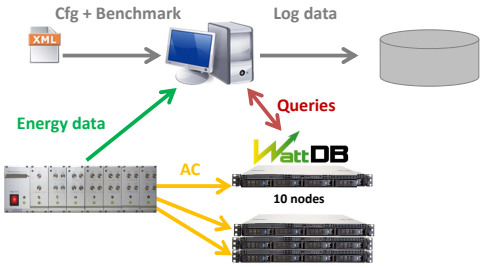


Fig. 5 Experimental measurement environment

a dedicated node with 25 Watts power consumption will require approximately 10 seconds and 250 Joules.

In order to save energy, reconfiguration overhead needs to pay off by reducing future query runtimes. Likewise, scale-in must trigger when the cluster is able to handle the workload with less nodes. To estimate the impact of reorganization, WattDB relies on a simplified cost model where upcoming workload predictions and maintenance costs are calculated.

In the following, we present key experiments using WattDB, starting from tests on the storage layer up to a fully dynamic, query-processing cluster.

3.3 Experimental Setup

Fig. 5 explains the setup used in all experiments. The bottom right depicts the cluster, currently consisting of 10 nodes. On the left, the measurement device, intercepting the power lines of the servers, is shown. All readings are sent to a dedicated computer, responsible for orchestrating benchmarks. This computer is sending queries to the cluster, measuring response times and throughput and correlates the data with power readings. All results are logged to a file for later analysis.

In the course of experimenting on the various WattDB layers, we varied the benchmarks and queries utilized to stress the system. To set experiments up to evaluate the cluster elasticity, we ran tests to determine the maximum workload the system was able to process. We refer to the workload intensity as *utilization*, which can be, for example, page requests per second or number of parallel queries. Hence, utilization will always be between 0%, i. e., no workload present, and 100%, i. e., the (predefined) maximum workload.

4 Dynamic Storage Allocation

In the course of examining the DB layers for their energy saving potential, we started with the bottom-most tier [13]. While the storage space of the cluster is physically accessible by all nodes and DB pages can be shipped over the network to remote nodes, logical restrictions, controlled by the master node, impose access limitations to defined areas of storage for each node. Hence, data placement can be easily altered and logical ownership of data can be dynamically reassigned to nodes, enabling flexible adaptation.

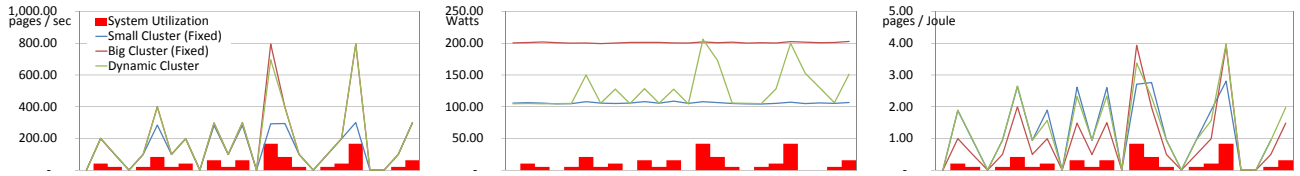
4.1 Experiments

These following experiments were conducted in 2012. At that time, we focused on the storage layer and WattDB did not yet have powerful query processing abilities. Therefore, we ran traces of OLTP page access patterns against the storage layer to test its performance.

The master node is responsible for power management by monitoring the IOPS of all disks in the cluster and triggering distribution of data blocks, when thresholds are crossed. Hence, if a storage node is overstressed with read/write requests, the data on disks is distributed to additional nodes, boosting total IOPS and bandwidth, but also increasing overall power use. Likewise, when disk IOPS fall below a threshold, the process is reversed and data is consolidated on fewer disks/nodes, reducing overall performance and energy.

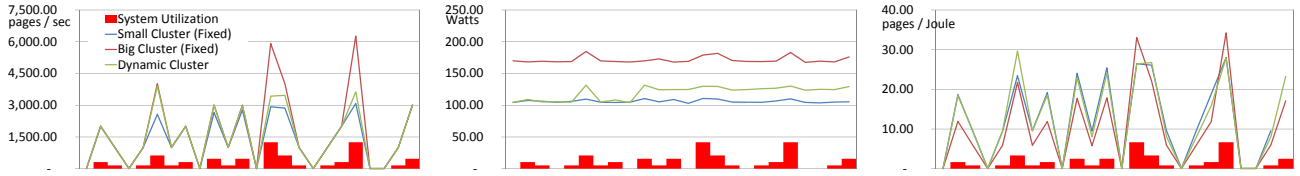
To make experiments on a dynamically adapting cluster comparable, we have repeated the same traces on six configurations: First, we distributed the data on two disks on a single storage node and disabled the power management to statically fix data distribution and number of nodes. The cluster consisted of the bare minimum of three nodes: one master, one processing node, and one storage node. Hence, this is the baseline as the least power-consuming, lowest performing configuration. Next, we repeated the trace on a cluster with the maximum number of nodes, one master, one processing node and five storage nodes. All data was statically allocated to all HDDs (10 in the cluster) and power management was again off. This run will determine peak performance and power use of our system.

After the baseline runs, we set up an unrestricted cluster, with power management on the master in full control of data distribution and nodes. We expect the



(a) Performance (b) Power consumption (c) Energy per query

Fig. 6 Benchmark results for an HDD cluster configuration



(a) Performance (b) Power consumption (c) Energy per query

Fig. 7 Benchmark results for an SSD cluster configuration

cluster to adapt to the current workload as the benchmark run proceeds. Finally, we replaced the five storage nodes having two storage disks each, with four storage nodes, each equipped with a single SSD, and repeated all three runs. Since SSDs promise much higher IOPS than HDDs, we increased the workload in these runs by a factor of 10 and still got decent response times.

4.2 Results

Fig. 6 and 7 summarize and compare our results for HDD- and SSD-based clusters, where the X-axis depicts the progress of the benchmark over time. Each graph consisting of three curves represent the two static clusters, small and big, and the dynamic cluster in between. At the bottom of all figures, we visualized the benchmark intensity varying between 0 and 100% of system utilization to facilitate the result interpretation.

Fig. 6(a) and Fig. 7(a) show the performance graphs for all runs, where IOPS are plotted on the Y-axis. As expected, the big HDD-based cluster delivers the most IOPS whereas the small cluster's IOPS is limited. At low utilization, hardly any difference is present; the small cluster falls behind only under high workloads, The dynamic cluster's performance is more or less identical to the big cluster. Only at peak utilization, maximum IOPS are reduced due to reconfiguration overhead.

Compared to HDDs, performance was better in all configurations for the SSD-based cluster. Yet, peak performance seems to be limited, even for the big static cluster, which may indicate another bottleneck in the system, e. g., CPU or network. Still, relative to each other, the three runs exhibit the same results as the first runs on HDDs and the dynamic cluster was able to almost match the big cluster's potential.

In Fig. 6(b) and 7(b), the power use in the course of the benchmark runs is plotted. As expected, the fixed configurations constantly draw the same amount of power, regardless of utilization. The dynamic cluster varies its size and, thus, exhibits changing power use between the baselines. As compared to HDDs, a variance of about 10 Watts is visible for the SSD-based cluster, even in the static configurations. For this reason, we conclude that SSDs seem to have a higher span between idle and full utilization. Further, the power consumption results reveal that not all nodes have been powered on automatically, even under high utilization (Fig. 7(b)). Because the SSDs were not fully utilized, there was no need to power-up additional nodes.

Combining performance and power consumption measurements, we can compute the energy efficiency (in terms of energy use per query) for all runs. Our results in Fig. 6(c) and 7(c) are expressed in pages per Joule. As expected, during low utilization, the smallest cluster exhibits the best energy efficiency. For more intense workloads, the small cluster is undersized and, with growing performance needs, energy efficiency declines. Therefore, at full utilization, the situation turns in favor of the big cluster.

The dynamic cluster automatically adjusts to the workload. Therefore, it delivers the same energy efficiency as the small cluster under low utilization. With rising workload, the cluster increases its size and, thus, starts to behave like the big cluster. Due to transition costs, its energy efficiency is slightly lower than that of the static cluster. The same observations hold, when replacing HDDs with SSDs, as plotted on Fig. 7(c). The small cluster handles low utilization well, whereas the

big cluster is suited for heavy workloads; the dynamic cluster combines the advantages of both.

Summary: With these experiments, we were able to verify our vision of a dynamic, energy-proportional storage layer that automatically adjusts to the workload by distributing DB pages. We identified the potential of SSDs to replace HDDs, because they provide more IOPS while consuming less power. Yet, redistributing data among nodes is a time-consuming task that cannot occur frequently. Instead, preparation for upcoming workloads might promise better adaptivity and bigger energy saving. Here, our scope was only the storage layer, acting as a page server to ship data to a processing node, where queries were evaluated. Such an approach heavily relies on the interconnecting network and storage nodes do only provide IOPS, not additional processing power to the cluster.

5 Dynamic Query Allocation

After evaluating the energy saving potential of a purely storage-centric approach, we have focused on WattDB's query layer in [12]. In the following experiments, we explore its energy saving potential. To test our hypothesis, we have implemented basic query optimization and evaluation techniques in WattDB, e. g., DB operators such as *IndexScan*, *HashJoin*, *Sort*, and *Aggregate*. Using these, we are able to answer OLAP queries on a pre-allocated dataset.

First, we ran simple query plans consisting of a few operators on different cluster configurations to estimate performance and power saving potential on the query layer. Our key idea was to offload query operators to remote nodes to free resources on the originating node and to distribute system utilization among more nodes to speed-up query processing. While rebalancing data is a time-consuming process, due to the cost of copying/moving pages, changing the mapping of query operators to nodes can be altered very quickly with immediate effects. On the downside, offloading introduces network delays, which might prevent better performance. Therefore, we conducted a series of experiments, quantifying the impacts of offloading.

Besides data access operators requiring local access to DB records, all other query operators can be placed on arbitrary nodes. To mitigate the negative effects of distribution, WattDB is using *vectorized volcano-style query operators* [6,4]; hence, operators ship a set of records on each call, instead of only passing a single record. This reduces the number of calls between operators and, in turn, network latencies. To further decrease network latencies, buffering operators are used to prefetch records from remote nodes, similar to buffering operators introduced by Zhou and Ross [22]. *Buffering operators* act as proxies between two (regular) operators; they asynchronously prefetch records, thereby hiding the delay of fetching the next set of records.

In WattDB, the query optimizer tries to put pipelining operators[3] on the same node to minimize latencies. Offloading pipeline operators to a remote node has little effect on workload balancing and, thus, does not pay off. In contrast, blocking operators, aka pipeline breakers[4], may be placed on remote nodes to equally distribute query processing. Blocking operators generally consume more resources (CPU, main memory) and are therefore good candidates for offloading and, hence, balancing utilization in the cluster.

5.1 Examining the Effects of Distribution

In Fig. 8, we show results of a micro-benchmark demonstrating the performance impact of distributing operators among nodes. The first (leftmost) run is a query containing a table scan locally running on a single node. The maximum throughput is slightly more than 40,000 records per second. In the next run, we added a local projection operator on top of the table scan, running on the same node. Although classic volcano-style operators ship only one record at a time, throughput is still high (approx. 34,000 records per second). To identify the influence of distribution, we ran the same operator combination on remote nodes. In this setting, throughput drops to less than 1,000 records per second, because each call to `next()` involves network delays. Next, we run the same operators on remote nodes with vectorized operators, where each call to `next()` returns a set of records at once. As a result, the operators need less calls to fetch all records and throughput increases to 24,000 records per second. In the last experiment, we included a buffering operator, which runs on the remote node and prefetches results from the underlying table scanner. While the projection operator is still processing a set of records, the buffer operator can asynchronously prefetch new records to further minimize network delays. In this constellation, throughput further increases to ~30,000 records per second.

Hence, with vectorized, volcano-style operators, network delays can be minimized to allow almost arbitrary operator placement in the cluster. Looking at these results, it is quite evident that distributing queries, instead of running all operators locally, is always a per-

[3] Pipelining operators can process one record at a time and emit the result, e. g., projection operators.

[4] Blocking operators need to fetch all records from the underlying operators, before emitting the first result record.

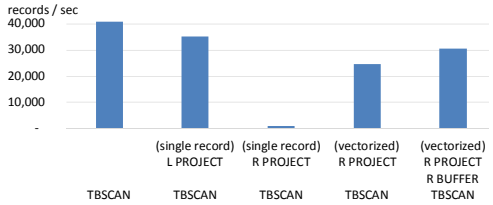


Fig. 8 Micro-benchmark testing throughput

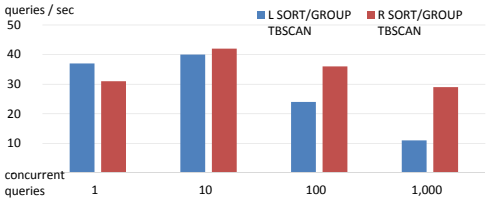


Fig. 9 Offloading queries, throughput

formance burden. Although we could reduce the negative impact, local query processing is still faster, compared to all other measurements. This is true for isolated queries, running on an underutilized node.

In a typical DBMS, multiple queries compete for resources like buffer space and CPU cycles. In these cases, offloading parts of the query plan to another node and, in turn, reducing the node's utilization may even improve performance. To verify that offloading query operators can increase overall query throughput, we have designed another micro-benchmark. For this experiment, we have run multiple queries concurrently, each consisting of a table scan with a subsequent sorting phase. In Fig. 9, throughput is shown for varying numbers of concurrent queries. The left (blue) bars plot it for a query plan, where both operators run on the same node. With increasing parallelism, throughput drops, because the node is overloaded and the queries compete for CPU and buffer. It increases when the sort operator is offloaded to another node (right (red) bars). Because of additional network communication, query throughput is initially lower than in the all-local case. With more concurrent queries, the additional buffer space and CPU power pay off, and throughput becomes substantially higher, compared to the single-node case. These experiments validate that distributing queries among nodes may increase overall performance, despite the additional network delays. Still, careful considerations have to be made regarding the network bandwidth and the nodes' utilization to estimate whether offloading will pay off. Also, offloading queries at low utilization levels is inferior to centralized processing.

5.2 Load-aware scheduler

We have implemented a cost-based scheduler to estimate operator costs during query optimization. With an additional framework, monitoring utilization of all nodes, we are able to dynamically place query operators on underutilized nodes and determine the number of processing nodes needed for the current workload. The power management component of WattDB is using this information to scale the cluster accordingly. In the following experiments, storage nodes were fixed, only the number of processing nodes was variable. Hence, the cluster was able to quickly power-up nodes, which were immediately able to participate in query processing. Also, powering-down was quickly possible, as soon as the last query on the node in question committed.

5.3 Experiments

We have generated the TPC-H dataset with a scale factor of 1 on a fixed number of storage nodes (between 1 and 7). The tables *LINEITEM* and *ORDERS* were partitioned and equally distributed among the storage nodes, whereas the smaller tables were replicated on all nodes. Because query capabilities were limited, we chose queries *Q1* and *Q4* for our benchmark.

Q1: This is an I/O- and CPU-intensive query, scanning through the LINEITEM table to select records predicated by date range. All matching records are then sorted, grouped, and aggregated. To leverage parallelization in our cluster, we split up the query into several parts. First, records are selected on the storage nodes and sorted locally. Results from all nodes are then forwarded to a processing node, where the presorted records are merged and aggregated.

Q4: This query requests records from ORDERS and LINEITEM, based on data ranges. Results are then joined and again sorted, grouped, and aggregated. As for Q1, evaluation is split among nodes. Storage nodes gather records from disk and pre-sort them, before a dedicated processing node runs the join, sort, group, and aggregation operators. In case of high utilization, some or even all of the last three steps in the query pipeline can also be offloaded to additional processing nodes.

First, we have run measurements on a fixed number of nodes to verify, whether or not performance and power use can be scaled as needed. We have run both queries (Q1 and Q4) separately at various intensities to generate several workloads between near-idle and full utilization. We have used the number of parallel DB clients to scale the workload in this experiments, hence, the more parallel queries, the higher the load.

After experimenting with fixed-sized clusters, we have enabled automatic load balancing on the master node and submitted a long-running benchmark with variable

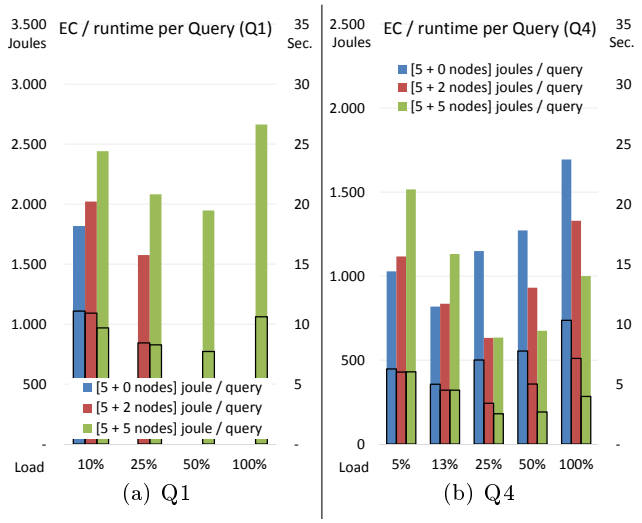


Fig. 10 Varying query load on 5 storage nodes and up to 5 processing nodes, using TPC-H Q1 (a) and Q4 (b)

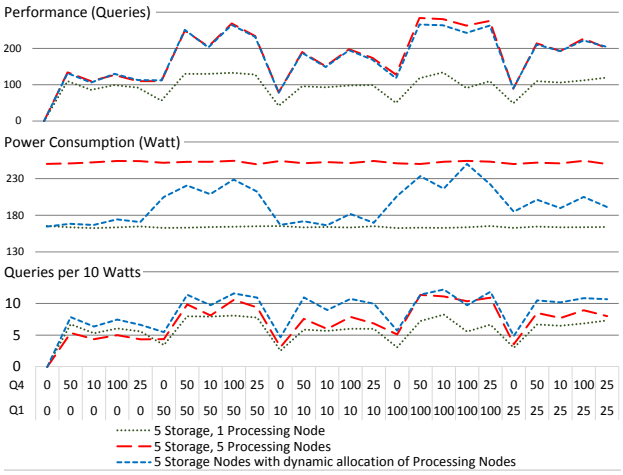


Fig. 11 Experiments on three cluster configurations

intensities. The benchmark uses a Q1/Q4 mix, stressing the DBMS with varying workloads. In this test, the number of *storage nodes* was fixed to five, while the active *processing nodes* could scale between one and five.

5.4 Results

In Fig. 10, we plot energy consumption and performance of the static runs of Q1 and Q4 respectively. On the X-axis, utilization is plotted, ranging from 5 to 100 %. The solid bars in the figure illustrate average energy consumption per query on the primary Y-axis, while the framed bars plot average runtime per query on the secondary Y-axis. For each specific utilization, we have run the same workload on different cluster configurations, varying the number of storage and processing nodes to reveal the dependency between cluster size, performance, and energy consumption.

As the results indicate, the best configuration in terms of energy use and performance depends on utilization: Lightweight workloads can be processed on a small cluster, while more intense loads require more powerful configurations. The small cluster in the previous experiments with *Q1* was not even able to process bigger workloads, i. e., handle a higher number of parallel DB clients. Further, the right balance between processing *and* storage nodes is important. With too many storage nodes, which cannot be simply turned off, idle power use is high, while too few storage nodes do not provide enough IOPS and, in turn, reduce query response times. Likewise, the number of processing nodes determines the system behavior. Because processing nodes do not hold persistent data, they can be turned on/off very quickly and, hence, it is not crucial to preselect the best-fitting amount for all workloads beforehand.

In Sect. 5.1, we have examined the performance impact of distributing query operators to remote nodes. With these results in mind, it is comprehensible how query processing benefits from adding processing nodes. Though these nodes have to request all data from storage nodes, they add main memory and CPU power to the cluster, thus relieving some load from the storage nodes, which, in turn, can focus on basic IO tasks. Especially OLAP applications require much analytical processing and benefit from pure processing nodes. Yet, as our experiments in the next sections suggest, a combination of storage and processing on the same node may be preferable to keep queries close to the data.

With insights from the fixed-size cluster benchmarks, we have set up a test with a variable number of DB clients over time. Thus, the workload changes between high and low utilizations. We have enabled WattDB's power management component to tune the number of processing nodes to the current needs. This component monitors query throughput and node utilization and scales the cluster out or in, respectively. Workloads in the experiment change every 600 seconds.

Fig. 11 plots the results in sequence. The X-axis depicts the number of parallel DB clients sending Q1 and Q4. On the Y-axis, performance, power consumption (and thus, cluster size), and energy efficiency are visualized. For comparison with the dynamic run, results include two static configurations with one and five processing nodes. These fixed runs mark the bounding box in which the dynamic cluster can adjust. The number of storage nodes for all three runs was fixed to five, since our previous experiments deemed this a good choice.

As the results show, the cluster is able to quickly adapt to changing workloads and exhibits almost the

same performance as the biggest static cluster. Yet, it consumes significantly less power at low and moderate workloads, because it powers down superfluous nodes.

Summary: With the previous experiments, we were able to test our implementation of a dynamic query processing layer for read-only workloads, on top of a static storage layer.

6 Dynamic Cluster

Previous research revealed power saving potential for a DBMS in both, storage and processing. Yet, combining both approaches to form a truly dynamic cluster, where the minimal configuration is a single node and the database can scale out based on the workload, may bring even more benefits. Therefore, we have redesigned our cluster to support scaling on both layers, storage and processing, simultaneously.

As our experiments on the storage layer revealed, data shipping is costly and, therefore, needs careful planning. At the same time, processing needs to run close to the data to mitigate remote data access. While previous experiments decoupled storage and processing to optimize their ensemble separately, we have re-engineered the nodes to support query execution and data storage in the following benchmarks. With this approach, data has to be partitioned logically to allow query optimizations like partition pruning and to keep processing close to the data. Further, we have included updating transactions to our workload mix to get another step closer to a fully-functional DBMS.

To integrate update transactions into WattDB without severely impacting query execution, we have implemented multi-version concurrency control (MVCC). With MVCC, updates create a new version of records, while older versions are still accessible for older transactions [3]. As another benefit compared to Multi-Granularity Locking with RX lock modes (MGL-RX), MVCC interoperates nicely with rebalancing on a logically partitioned database. To resize a partition, primary-key ranges are simply removed from the source and inserted into the target partition with a new version identifier. The whole process works under MVCC and therefore guarantees atomicity and isolation [14].

6.1 Experiments

To test both, OLTP and OLAP on our cluster, we have deployed the TPC-H dataset (scale factor 100) on the cluster and run two workloads on the data. For OLAP, we have used standard TPC-H queries on the dataset. For OLTP, we have created TPC-C-like queries to run on the TPC-H dataset, enabling to run both, OLTP and OLAP in parallel on the same data. With both query types, we were able to run a realistic workload mix.

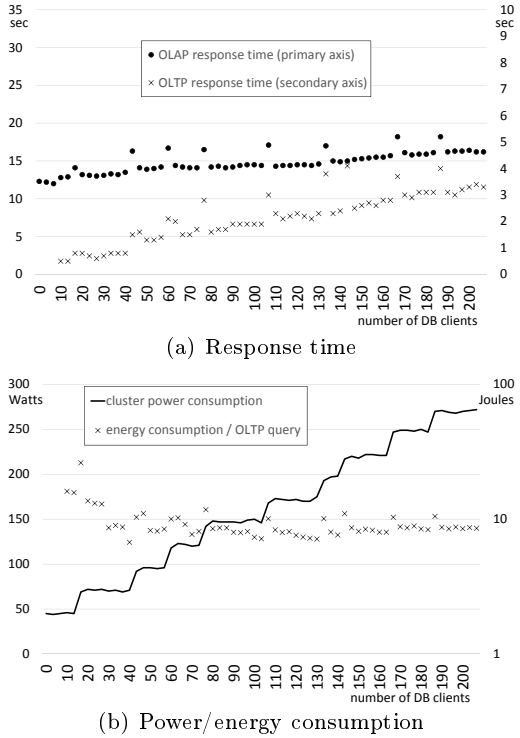


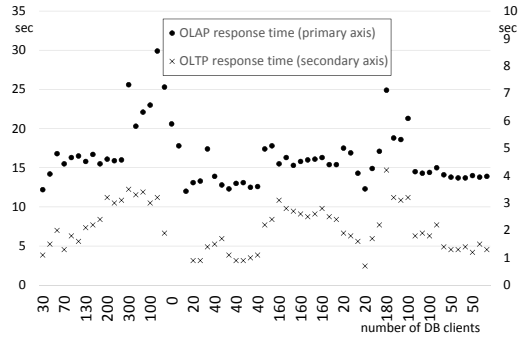
Fig. 12 Increasing load on a dynamic cluster

First, we have run a workload with steadily rising intensity to test the adaptivity (scale-out) of our cluster to all utilizations, from low to high. During each of the runs, a single DB client was sending OLAP queries to the database and reported their response time. The number of parallel OLTP clients was variable and used as a workload driver to trigger utilization.

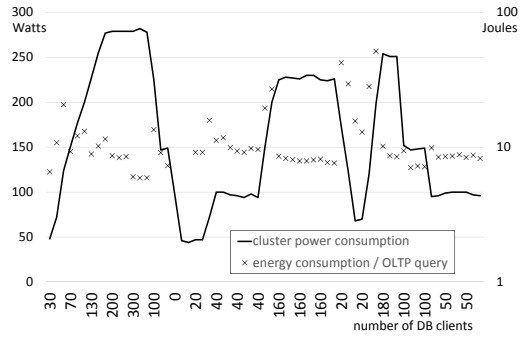
Next, to test the elasticity of our implementation, we have generated a workload with variable utilization as described earlier and benchmarked the cluster against it. Again, one DB client sent OLAP queries while the number of OLTP clients was variable.

6.2 Results

We have run the scale-out benchmark on an unprepared cluster. Hence, as soon as query workload increased, the DBMS had to scale out and redistribute data. Fig. 12 visualizes response time and energy efficiency, where the X-axis indicates the number of OLTP DB clients, ranging from 0 to 200 and stressing the database. We plot response times for OLAP on the primary Y-axis and for OLTP on the secondary Y-axis (Fig. 12(a)).



(a) Response time



(b) Power/energy consumption

Fig. 13 Dynamic cluster supported by forecasting

Fig. 12(b) shows power consumption and OLTP query energy efficiency for each utilization level.

Starting with a low utilization, the database is running on a single node only, keeping the other 9 nodes suspended. As a consequence, (all partitions of) the entire dataset had to be allocated on the node's disks. Power use, as plotted in Figure 12(b), is initially low (about 45 Watts for the whole cluster). With increasing utilization, WattDB dynamically wakes up nodes and assigns DB partitions to them. Hence, re-partitioning, as previously described, needs to physically move records among nodes. Therefore, in parallel to the query workload, the movement takes up additional system resources. While the database is re-configuring, avg. query runtimes increase by 3 seconds for OLTP workloads and up to 6 seconds for OLAP workloads because of the extra work. Moving data among nodes takes approximately 30 to 120 seconds, depending on the amount of data to be moved. The spikes in Figure 12(a) visualize the degraded performance while moving data partitions.

Because repartitioning is costly and a cluster already under high utilization performs worse when repartitioning [14], we implemented a simple forecasting approach to give the cluster time to react before workloads change. Therefore, we have run the workload mix on a "proactive" cluster, which we have continuously informed about the three following workloads, i. e., the requesting DB clients in the next minutes. This knowledge is used by the master to proactively partition data to the needs of the future workloads.

Fig. 13 plots the results from the runs on a proactive cluster. One one hand, query response times are more fluctuating, compared to our scale-out experiment, since the DBMS has to constantly adjust its size to the workload. Therefore, power consumption is also higher, because nodes are turned on in anticipation of the workloads. But the cluster is adjusting quite well to workload changes and scales in and out as expected. Yet, overall response times are higher due to rebalancing overhead. A more flexible, easier to alter partitioning scheme may better support our dynamic approach.

Furthermore, running OLAP in parallel to OLTP queries takes a huge toll on our system. Query response times for OLTP are very high (up to 4 seconds per query) when running in parallel with OLAP. Therefore, we have decided to separate workloads in the future to examine effects on both query types separately.

7 Partitioning

Our first experiments, examining the storage layer, were shipping data segments to remote nodes to balance workloads. Since this approach operates on physical data blocks with no interaction on the query execution layer, we call it *physical partitioning*. It has proven to be an easy to manage and simply to achieve way of repartitioning data in our cluster. Yet, the approach is limited to a storage-centric view without leveraging knowledge about the data stored inside the segments.

Therefore, we have implemented MVCC and a more sophisticated partitioning scheme, supporting the transfer of logical ownership of the data among nodes. With MVCC, records are moved between logical DB partitions, which involves updating mapping information in the query execution layer. Therefore, we call this approach *logical partitioning*. While logical partitioning integrates well with dynamic balancing, and both, storage and query workloads can be tuned with this approach, look-up overhead for records to be moved is very high, compared to physical partitioning. Since the query execution layer is involved, a lot of effort has to be done to guarantee ACID-compliant movement of records, including locking and proper version control.

As mentioned, both approaches exhibit benefits, but also show serious drawbacks. Therefore, a combined approach, balancing storage and processing is needed, leveraged by a flexible and dynamic partitioning scheme. We have extended *physiological partitioning* [18] to partition data among nodes, not CPU cores (as in the original design). Similar to the original approach, we encap-

sulate key ranges in partitions and assign them exclusively to eliminate contention. In our implementation, partitions still consist of segments, but each segment now keeps a primary-key index for all records within it, forming a self-contained sub-partition. The partition only contains a small index on top, keeping information about key ranges in all segments attached.

Rebalancing data is easy with physiological partitioning, similar to physical partitioning: Before moving a segment, some meta-data on the partition needs to be updated to inform queries of the imminent rebalancing. Next, the segment is locked read-only and updating transaction need to finish their changes. Now, the whole segment can be moved to another node, followed by another meta-data update. Finally, query optimization can regard the changed partitioning and queries can access the records on the new node [16]. To mitigate overhead, additional nodes can support the cluster while data movement is in progress. These helper nodes are powered on for a short time to provide additional DB buffer pages over rDMA[5] and handle logging for the nodes involved in the move. Though additional nodes increase power use, their extra performance will support the cluster and reduce query runtimes.

7.1 Experiments

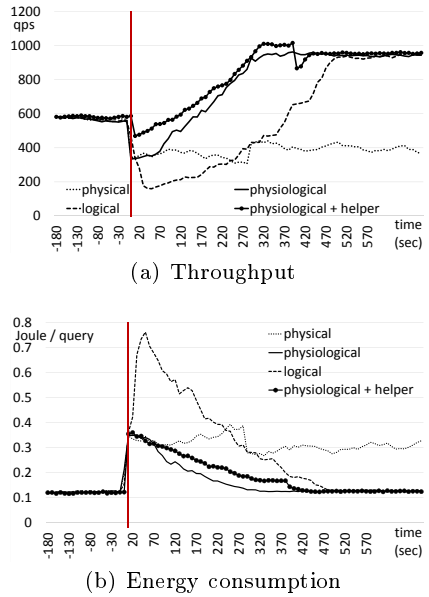
To compare all three partitioning schemes, we have measured query response times and energy consumption while rebalancing using each of the approaches. We have deployed a TPC-C dataset with a scale factor of 1,000, hence, about 200 GB of raw data was stored on the cluster. Similar to previous experiments, we have deployed a number of parallel DB clients, querying the database, while we triggered repartitioning.

7.2 Results

In Fig. 14, we depict results from four runs of repartitioning. The vertical line in the graphs marks the start of repartitioning, the X-axis denotes the time relative to that. The upper graph plots query throughput per second, while the lower graph shows energy consumption per query. All runs, regardless of partitioning scheme initially exhibit the same performance and power consumption, as they are running on identical cluster configurations. After triggering a scale-out from two to four nodes at $t = 0$, results reveal differences.

All approaches exhibit decreased performance right after rebalancing kicks in. Throughput under *Physical*

[5] rDMA = remote direct memory access



(a) Throughput

(b) Energy consumption

Fig. 14 Benchmark results for various partitioning schemes

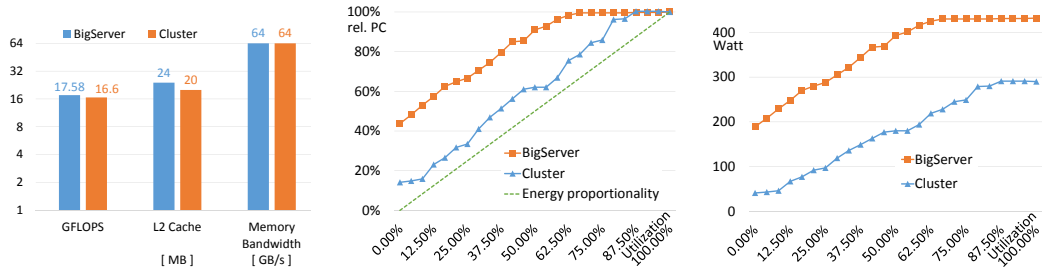
partitioning declines slightly from ~600 tps to 400 tps and never recovers. Likewise, energy use per query gets higher. Because physical partitioning only moves data segments to remote nodes, page accesses to remote segments now involves network communication overhead, which severely impacts query performance.

Logical partitioning shows an initially strong decline in throughput, but after a few minutes, query throughput increases significantly. Similarly, energy efficiency improves. With logical partitioning, records, based on primary key ranges, are moved to nodes, which requires look-up overhead. This explains the initial decline in performance. With more and more records now staying in another partition, the additional nodes will take over evaluating queries on these and thus, add processing power, which increases throughput.

Finally, *physiological partitioning* combines both approaches and only suffers from a smaller initial performance decline like physical partitioning, but quickly recovers to gain performance like logical partitioning. With two additional helper nodes, supporting the nodes under rebalancing as previously described, the performance impact of repartitioning can be further mitigated, with the cost of higher power consumption and, therefore, worse energy efficiency. After the rebalancing completed, the helper nodes were switched off again.

8 Comparison with a Brawny Server

Our findings reveal potential for saving energy by deploying a scale-out cluster of lightweight nodes. Yet, a distributed DBMS needs coordination and may exhibit



(a) Theoretical performance (b) Relative power consumption (c) Power consumption

Fig. 15 Power consumption and performance figures for both systems

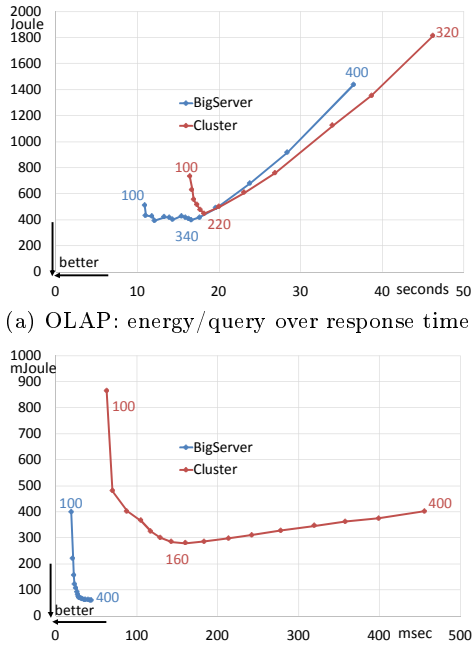
serious performance drawbacks, compared to a centralized server. In the following, results from our benchmarks compare the cluster to a brawny server [17]. To make a fair comparison, we have selected a server with two Intel Xeon X5670 processors, 24 GB of DRAM. Further, we have reduced the storage disks in the cluster to 10 SSDs, to match the maximum number of disks the brawny server can handle. The two configurations provide roughly the same number of GFLOPS, L2 cache sizes, and memory bandwidth as depicted in Fig. 15(a). Fig. 15(c) plots both systems' power consumption. The cluster draws about 280 Watts at maximum, whereas the big server consumes over 400 Watts.[6] In Fig. 15(b), power consumption relative to their maximum is shown. It is already obvious that the cluster is more energy proportional than the big server.

8.1 Experiments

To compare performance and elasticity of both systems, we have run a series of benchmarks on both systems. First, we have run performance-centric benchmarks to estimate the peak load both systems can handle. We repeated the tests with varying DB clients to establish a saturation point, i. e., how many parallel queries the systems can handle. Second, we have chosen a workload mix of variable intensities as already described. We have run separate OLTP and OLAP workloads to dissect the systems' behavior on both individually.

8.2 Results

In Fig. 16, results from performance-centric runs are depicted. The first figure plots both systems' OLAP query runtime (X-axis) versus energy use per query (Y-axis). The numbers on the data points annotate the number of parallel DB clients. From the figure, we can



(a) OLAP: energy/query over response time

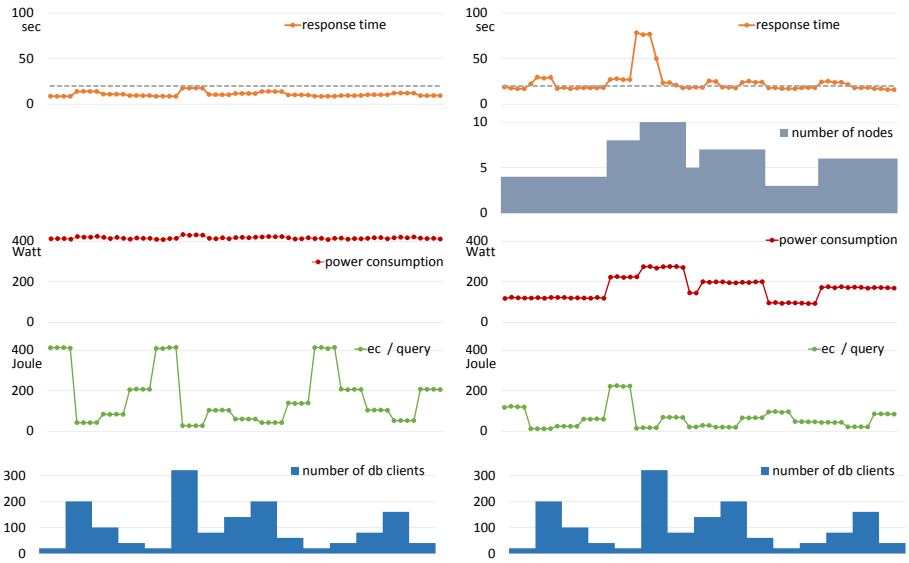
(b) OLTP: energy/query over response time

Fig. 16 Peak Performance and energy consumption

conclude that the big server handles queries generally faster than the cluster, while exhibiting better energy efficiency. The cluster handles medium-sized workloads almost as well as the big server, yet more than 220 parallel clients seem to overload the cluster as runtimes and energy consumption quickly increase.[7] In the next figure, the same experiments using OLTP workloads are shown. Here, the big server exhibits far better performance and energy efficiency than the cluster. Further, query response times on the brawny server increase only slightly with the number of DB clients and the system does not show any signs of saturation. Consequently, energy efficiency improves continuously. The cluster is saturated with 160 clients; heavier workloads result in even higher runtimes and energy consumption.

[6] Note that 100% utilization is not equivalent in both systems, the big server is possibly much faster than the cluster.

[7] We also noticed, that OLTP runtimes are significantly lower compared to our experiments with OLAP in parallel.



(a) Results on the big server (b) Results on a forecasting cluster

Fig. 17 Dynamic OLAP workload on the big server and the cluster

These results reveal that the cluster seems to be better fitted for OLAP workloads. Considering the access patterns of both query types, the differences are explainable: OLAP queries read huge amounts of records, join and aggregate them to satisfy analytical inquiries. Hence, this type of query can be parallelized very easily, leveraged by the high number of independent nodes in the cluster. Further, OLAP does not need synchronization, since analytics are read-only. In turn, OLTP queries are very selective, touching a few records frequently. Since a lot of these transactions update records, concurrency control is needed, requiring synchronization among all cluster nodes to keep data consistent.

After evaluating peak performance, we ran experiments with real-world usage patterns to test both systems's behavior under average conditions. We expect the big server to exhibit far less elasticity than the cluster, since it cannot rebalance. Yet, the limiting factor for dynamic repartitioning is migration cost, i. e., the performance impact and time it takes to move data between nodes. Workloads in the dynamic experiment change every 5 minutes, and—based on our previous findings about the importance of preparation—we notified the systems in advance of the 6 upcoming workloads. As stated by Kramer et. al. [8], workloads are often repetitive and, therefore, quite easy to forecast.

In Fig. 17, results from OLAP runs on both systems are graphed. The first graphs plot response times of both systems. As expected, the big server does not exhibit much variance in runtimes, since it is instantly ready and does not need reconfiguration. Query runtimes are therefore fast and always beat the target response times. Yet, power consumption is always high, resulting in relatively high energy use per query.

The plots on the right side of Fig. 17 illustrate results from the same workload on a dynamic cluster using forecasting. While query runtimes are a bit worse compared to the big server, target response times are met mostly. The second subfigure shows the number of running nodes. For the big server, we omitted the figure for obvious reasons. The cluster is permanently rebalancing to match the worst-case expected workload in the near future, which results in constant node fluctuation. In comparison with the brawny server, the cluster is a little slower, but consumes far less energy, making it the more energy-efficient choice.

We repeated similar OLTP experiments [17] and revealed that the cluster is less suited for such workloads.

9 Conclusion

In this article, we have summarized our findings over the last years. We started our explorations on the storage layer, where we implemented a dynamically balancing, storage-centric DB cluster. Although we have shown that optimizations on the storage lead to better energy efficiency, high access latencies and limited scalability prevent bigger savings. Next, we implemented a dynamic query execution layer, where the number of nodes participating in query processing scales with the workload. By running a varying number of nodes, we could show that it is also possible to save energy at this database layer. By combining both approaches, data storage and query execution, in WattDB to form a truly

dynamic cluster, scaling out and back in as needed, our vision of an energy-proportional DBMS has come true.

10 Acknowledgments

The research project *Energy-Efficient Processing in Database Systems* was partly funded by the German Research Foundation (DFG) under the contracts HA 1286/8-1 and HA 1286/8-2. Finally, we would like to thank the anonymous reviewers for their insightful reviews.

References

1. Albers, S.: Energy-efficient Algorithms. Commun. ACM **53**(5), 86–96 (2010)
2. Barroso, L.A., Hölzle, U.: The Case for Energy-Proportional Computing. IEEE Computer **40**(12), 33–37 (2007)
3. Bernstein, P.A., Goodman, N.: Multiversion Concurrency Control—Theory and Algorithms. ACM Trans. Database Syst. **8**(4), 465–483 (1983)
4. Boncz, P.A., Zukowski, M., Nes, N.: MonetDB/X100: Hyper-Pipelining Query Execution. In: CIDR Conf., pp. 225–237 (2005)
5. Genzmer, C., Hudlet, V., Park, H., Schall, D., Senellart, P.: The SIGMOD 2010 Programming Contest - A Distributed Query Engine. SIGMOD Record **39**(2), 61–64 (2010)
6. Graefe, G.: Volcano—An Extensible and Parallel Query Evaluation System. IEEE Trans. on Knowl. and Data Eng. **6**(1), 120–135 (1994)
7. Härder, T., Rahm, E.: Hochleistungs-Datenbanksysteme – Vergleich und Bewertung aktueller Architekturen und ihrer Implementierung. IT **29**(3), 127–140 (1987)
8. Kramer, C., Höfner, V., Härder, T.: Lastprognose für energieeffiziente verteilte DBMS. Proc. 42. GI-Jahrestagung 2012, LNI 208 pp. 397–411 (2012)
9. Lang, W., Harizopoulos, S., Patel, J.M., Shah, M.A., Tsirogiannis, D.: Towards Energy-Efficient Database Cluster Design. PVLDB **5**(11), 1684–1695 (2012)
10. Lang, W., Patel, J.M.: Towards Eco-friendly Database Management Systems. In: CIDR Conf. (2009)
11. Poess, M., Nambiar, R.O.: Energy Cost, The Key Challenge of Today's Data Centers: A Power Consumption Analysis of TPC-C Results. PVLDB **1**(2), 1229–1240 (2008)
12. Schall, D., Härder, T.: Energy-Proportional Query Execution Using a Cluster of Wimpy Nodes. In: SIGMOD Workshops, DaMoN, pp. 1:1–1:6 (2013)
13. Schall, D., Härder, T.: Towards an Energy-Proportional Storage System using a Cluster of Wimpy Nodes. In: BTW Conf., LNI 214, pp. 311–325 (2013)
14. Schall, D., Härder, T.: Approximating an Energy-Proportional DBMS by a Dynamic Cluster of Nodes. In: DASFAA Conf., LNCS 8421, pp. 297–311 (2014)
15. Schall, D., Höfner, V., Kern, M.: Towards an Enhanced Benchmark Advocating Energy-Efficient Systems. In: TPCTC Conf., LNCS 7144, pp. 31–45 (2012)
16. Schall, D., Härder, T.: Dynamic Physiological Partitioning on a Shared-nothing Database Cluster. CoRR **abs/1407.0120** (2014)
17. Schall, D., Härder, T.: Energy and Performance—Can a Wimpy-Node Cluster Challenge a Brawny Server? CoRR **abs/1407.0386** (2014)
18. Tözün, P., Pandis, I., Johnson, R., Ailamaki, A.: Scalable and Dynamically Balanced Shared-Everything OLTP with Physiological Partitioning. VLDB J. **22**(2), 151–175 (2013)
19. Tsirogiannis, D., Harizopoulos, S., Shah, M.A.: Analyzing the Energy Efficiency of a Database Server. In: SIGMOD Conf., pp. 231–242 (2010)
20. Wang, J., Feng, L., Xue, W., Song, Z.: A Survey on Energy-Efficient Data Management. SIGMOD Record **40**, 17–23 (2011)
21. Willhalm, T.: Intel Performance Counter Monitor - A Better Way to Measure CPU Utilization. Tech. rep., Intel Corporation (2012)
22. Zhou, J., Ross, K.A.: Buffering Database Operations for Enhanced Instruction Cache Performance. In: SIGMOD Conf., pp. 191–202 (2004)