Approximating an Energy-Proportional DBMS by a Dynamic Cluster of Nodes

Daniel Schall and Theo Härder

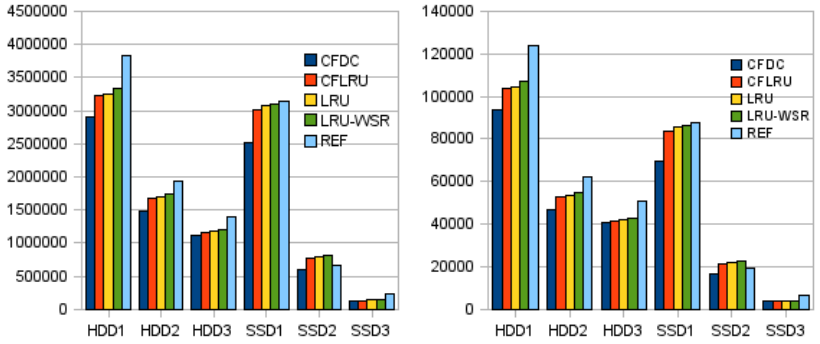
DBIS Group, University of Kaiserslautern, Germany
{schall,haerder}@cs.uni-kl.de

Abstract. The most energy-efficient configuration of a single-server DBMS is the highest performing one, if we exclusively focus on specific applications where the DBMS can steadily run in the peak-performance range. However, typical DBMS activity levels—or their average system utilization—are much lower and their energy use is far from being *energy proportional*. Built of commodity hardware, *WattDB*—a distributed DBMS—runs on a cluster of computing nodes where energy proportionality is approached by dynamically adapting the cluster size. In this work, we combine our previous findings on energy-proportional storage layers and query processing into a single, transactional DBMS. We verify our vision by a series of benchmarks running OLTP and OLAP queries with varying degrees of parallelism. These experiments illustrate that WattDB dynamically adjusts to the workload present and reconfigures itself to satisfy performance demands while keeping its energy consumption at a minimum.

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. Due to their narrow power spectrum between idle and full utilization [1], 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.

Energy proportionality describes the ability of a system to reduce its power consumption to the actual workload, i.e., a system, delivering only 10% of its peak performance, must not consume more than 10% of its peak power. That goal could be approached by exploiting hardware-intrinsic properties, e.g., CPUs *automatically* entering sleep states or hard disks spinning down when idle. Unfortunately, current hardware is not energy proportional. For example, DRAM chips consume a constant amount of power—regardless of their use—and it is not possible to turn off unused memory chips in order to reduce energy consumption. Spinning down hard disks when idle conflicts with long transition times and results in slow query evaluation.



(a) Execution times (ms) **(b)** Energy use (J)

Fig. 1: Performance and energy consumption running the TPC-C trace

1.1 Energy Efficiency Limits of Single-Server DBMSs

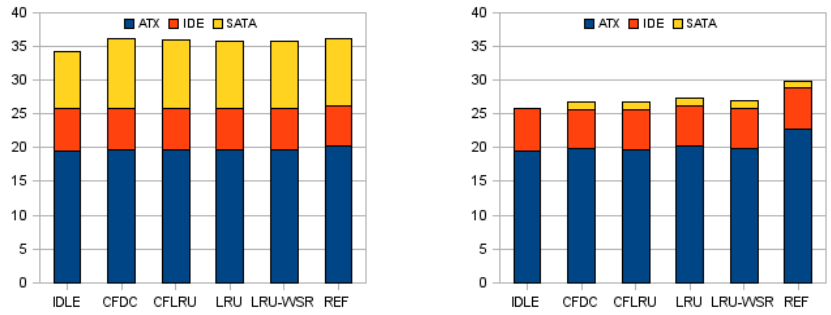
Using a key result of experiments with a single-server DBMS [10], we want to illustrate the close linkage of execution times and energy efficiency. All experiments were conducted in an identical system setting, i. e., ATX, IDE, memory size, OS, DBMS (except buffer management), and workload were left unchanged. For this reason, the details are not important in this context. Our goal was to reveal the relationship concerning performance and energy use for different external storage media[1] and buffer management algorithms[2].

To represent a realistic application for our empirical measurements, we recorded an OLTP trace (a buffer reference string using a relational DBMS) of a 20-minutes TPC-C workload with a scaling factor of 50 warehouses. The test data (as a DB file) resided on a separate magnetic disk/SSD (data disk, denoted as SATA). The data disks, connected one at a time to the system, represent *low-end* (HDD1/SSD1), *middle-class* (HDD2/SSD2), and *high-end* (HDD3/SSD3) devices. Execution times and related energy use in Figure 1 are indicative for what we can expect for single-server DBMSs by varying the storage system configurations. The storage device type dominantly determines execution time improvement and, in turn, reduction of energy use. In our experiment, the algorithmic optimizations and their relative influence to energy efficiency are noticeable, but less drastic.

The key effect identified by Figure 1 is further explained by Figure 2, where the break-down of the average working power of hardware components of interest

[1] We used magnetic disks (HDD1: 7.200 rpm, 70 IOPS; HDD2: 10.000 rpm, 210 IOPS; HDD3: 15.000 rpm, 500 IOPS) and flash storage (read/write IOPS) (SSD1: 2.700/50, SSD2: 12.000/130, SSD3: 35.000/3.300).

[2] CFDC [9] optimizes page caching for SSDs. Here, we cross-compared CFDC to LRU, CFLRU [11], LRU-WSR [7], and REF [15], some of which are also tailor-made for SSD use.



(a) HDD3 **(b)** SSD3

Fig. 2: Break-down of average power (W)

is compared with their *idle* power values. The figures shown for HDD3 and SSD3 are indicative for all configurations; they are similar for all devices, because ATX—consuming the lion's share of the energy—and IDE remained unchanged. Ideally, utilization should determine the power usage of a component. But, no significant power variation could be observed when the system state changes from idle to working or even to full utilization. Because the time needed to run the trace is proportional to the energy consumption, the fastest algorithm is also the most energy-efficient one. This key observation was complemented by [17] with a similar conclusion 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*".

In summary, energy saving is impressive, if we consider the experiment in isolation where system utilization is steadily kept very high, i. e., $> 90\%$. However, typical servers mostly reach an average CPU utilization of only $\sim 30\%$ and often even less than $\sim 20\%$ [2].

1.2 Varying DBMS Service Needs

We have shown in [14] that real-world workloads usually do not stress DB systems 24/7 with peak loads. Instead, the workloads alternate in patterns between high and near-idle utilization. But, DB systems have to be tailored to the peak performance to satisfy incoming queries and potential users. Therefore, DB servers usually come with big DRAM buffers and a number of disks as external storage—both components that consume a lot of energy. The majority of these resources is only needed in rare time intervals to handle peak workloads. All other times, they lie waste, thereby substantially decreasing the overall energy efficiency of the server. During times of underutilization, overprovisioned components are not needed to satisfy the workload. By adjusting the DB systems to the current workload's needs, i.e., making the system energy proportional, energy usage could be lowered while still allowing the maximum possible performance.

Both observations sketched above have strongly guided the design of WattDB: Section 2 outlines its cluster hardware, the related power consumption, and the most important aspects of its software design. In Section 3, we introduce our experimental setup, before we discuss the results of our empirical benchmark runs in Section 4. Finally, we conclude our results and give an outlook to our future work in Section 5.

2 The WattDB Approach

Based on the findings outlined above, we concluded that a single-server-based DBMS will never be able to process real-world workloads in an energy-efficient way. A cluster of lightweight (wimpy) nodes is more promising, as nodes can be dynamically switched on or off, such that the cluster can be adjusted to the current workload. Lang et al. [8] have shown that a cluster suffers from "friction losses" due to coordination and data shipping overhead and is therefore not as powerful as a comparable heavyweight server. On the other hand, for moderate workloads, i. e., the majority of real-world database applications, a scale-out cluster can exploit its ability to reduce or increase its size sufficiently fast and, in turn, gain far better energy efficiency.

In [12], we already explored the capabilities and limitations of a clustered storage architecture that dynamically adjusts the number of nodes to varying workloads consisting of simple *read-only page requests* where a large file had to be accessed via an index[3]. We concluded that it is possible to approximate energy proportionality in the storage layer with a cluster of wimpy nodes. However, attaching or detaching a storage server is rather expensive, because (parts of) datasets may have to be migrated. Therefore, such events (in appropriate workloads) should happen on a scale of minutes or hours, but not seconds.

In [13], we have focused on the query processing layer—again for varying workloads consisting of two types of *read-only SQL queries*—and drawn similar conclusions. In this contribution, we revealed that attaching or detaching a (pure) processing node is rather inexpensive. Hence, such an event can happen in the range of a few seconds—without disturbing the current workload too much.

In this paper, we substantially extended the kind of DBMS processing supported by WattDB to *complex OLAP / OLTP workloads consisting of read-write transactions*. For this purpose, we refined and combined both approaches to get one step closer to a fully-featured DBMS. Opposed to our previous work, we treat all nodes identical in this paper; hence, all nodes will act as storage and processing nodes simultaneously.

2.1 Cluster Hardware

Our cluster hardware consists of n (currently 10) identical nodes, interconnected by a Gigabit-ethernet switch. Each node is equipped with an Intel Atom D510

[3] Starting our WattDB development and testing with rather simple workloads facilitated the understanding of the internal system behavior, the debugging process, as well as the identification of performance bottlenecks.

CPU, 2 GB DRAM and three storage devices: one HDD and two SSDs. The configuration is considered Amdahl-balanced [16], i. e., balanced between I/O and network throughput on one hand and processing power on the other. By choosing commodity hardware with limited data bandwidth, Ethernet wiring is sufficient for interconnecting the nodes. All nodes are connected to the same Ethernet segment and can communicate with each other.

2.2 Power Consumption

As we have chosen lightweight nodes, the power consumption of each node is rather low. A single node consumes ~22 - 26 Watts when active (based on utilization) and 2.5 Watts in standby mode. The interconnecting switch needs 20 Watts and is included in all measurements.

The minimal configuration of the cluster consists of a single node, called the *master node*. All functionalities (storage, processing, and coordination) can be centralized using only a single node, thereby minimizing the static energy usage. In case, all nodes are running, the overall energy use of the cluster reaches ~270 Watts. This is another reason for choosing commodity hardware which uses much less energy compared to server-grade components. For example, main memory consumes ~2.5 Watts per DIMM module, whereas ECC memory, typically used in servers, consumes ~10 Watts per DIMM. Likewise, our HDDs need less power than high-performance drives, which makes them more energy efficient.

2.3 Software Design

A single wimpy node can quickly become a hotspot which may slow down query processing of the entire cluster. To mitigate bottlenecks, *WattDB* is designed to allow dynamic reconfiguration of the storage and query processing layers.

The *master node* accepts client connections, distributes incoming queries, and administrates metadata for all cluster nodes. This node is also responsible for controlling the power consumption of the cluster by turning nodes on and off. However, the master is not different from the rest of the nodes; it is also able to process queries and manage its own storage disks.

Storage Structures and Indexes In WattDB, data is stored in tables, which are subdivided into partitions. Each partition is organized as a heap and consists of a set of segments, where a segment specifies a range of pages on a hard drive. Physical clustering of pages is guaranteed inside each segment. To preserve locality of data, segments are always assigned to disks on the same node managing the partition. Indexes implemented as B*-trees can be created within a partition to speed up query evaluation.

Dynamic Partitioning Scheme From a logical point of view, database tables in WattDB are horizontally sliced into *partitions* by primary-key ranges. Each partition is assigned to a single node, possibly using several local hard disks. This node is responsible for the partition, i. e., for reading pages, performing projections and selections, and for propagating modified pages while maintaining isolation and consistency.

To support dynamic reorganization, the partitioning scheme is not static. Primary-key ranges for partitions can be changed and data can migrate among partitions on different nodes to reflect the new scheme. Hence, partitions can also be split up into smaller units, thereby distributing the data access cost among nodes, and can be consolidated to reduce its storage and energy needs.

Partitioning information is kept on the master node in an ordered, unbalanced tree to quickly identify partitions needed for specific queries. Pointers reference either inner nodes with further fragmentation of the primary-key range or point to a partition where the data is stored. Note, while moving or restructuring of a partition is in progress, its old and new state must be reachable. Therefore, each pointer field in the tree can hold two pointers. While a partition is reorganized (split or merged), the pointers may point to two different partitions: In case of a split, the first pointer refers to the new partition to which a writing transaction copies the corresponding records, whereas the second pointer references the old partition where non-moved records still remain (and vice versa in case a merge is in progress). An appropriate concurrency control scheme should enable reads and updates of the new and old partition while such a reorganization is in progress.

Figure 3 plots three stages of an exemplary partition tree while a split is processed. The first stage shows (a fraction of) the initial key distribution. In the second stage, the key range between 1,000 and 2,000 contained in partition 3.4 is split into two partitions, where a new partition 3.6 has to be created. A transaction scans the old partition 3.4 and moves records with keys between 1,000 and 1,500 to the new partition 3.6. Records with primary keys above 1,500 stay in the old partition. To allow concurrent readers to access the records, the read pointer (marked r) still points to the old partition. Thus, reading transactions will access both, the new and old partition to look for records. The write pointer already references the new partition, redirecting updates to the new location. In the last stage, the move operation is assumed to have succeeded, and read and write pointer both reference the new partition. Moving an entire partition can be considered as a special case of such a split.

By keeping records logically clustered, the query optimizer on the master node can quickly determine eligible partitions and distribute query plan operators to run in parallel. At query execution time, no additional look-ups have to go through the master node.

Concurrency Control As every other DBMS, WattDB needs to implement mechanisms for concurrency control to ensure ACID properties [5]. Therefore, access to records needs to be coordinated to isolate read/write transactions. When changing the partitioning schema and moving records among partitions, concurrency control must also coordinate access to the records in transit. Classical pessimistic locking protocols block transactions from accessing these records until the moving transaction commits. This leads to high transaction latency, since even readers need to wait for the move to succeed. Furthermore, writers must postpone their updates to wait for the move to terminate.

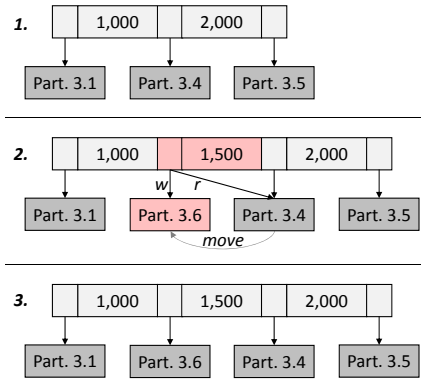


Fig. 3: Three steps of a split in the partition tree with updates of r/w pointers

Multiversion Concurrency Control (MVCC) allows multiple versions of database objects to exist. Each modification of data creates a new version of it. Hence, readers can still access old versions, even if new transactions changed the data. Each data element keeps a version counter of its creation and deletion date to enable transactions to decide which records to read by comparing the version information with the transaction's own version counter. While concurrent writers still need to synchronize access to records, readers will always have the correct version and will not get blocked by writers [3]. The obsolete versions of the records need to be removed from the database from time to time by a process we call *garbage collection* (GC).[4]

While MVCC allows multiple readers lock-free access to records, writing transactions still need to synchronize access with locks. Hence, deadlocks can arise where two or more transactions wait for each other and none can make any progress. Therefore, each node has a deadlock detection component that keeps track of locally waiting transactions in a wait-for graph (see [4]). To detect deadlocks spanning multiple nodes, a centralized detector on the master aggregates information of the individual nodes to create a global wait-for graph.

Move Semantics Transactions need to be ACID compliant; hence, moving records among partitions must also adhere to these properties. Therefore, it is vital to move the records inside a dedicated transaction acting as follows:

1. Update the partition tree information with pending changes
2. Read records from the source partition
3. Insert the records into the target partition
4. Delete the records from the source[5]
5. Update the partition tree information with final changes
6. Commit

[4] PostgreSQL calls it *vacuum*.
[5] Note that the order of Insert and Delete is not important.

Because the movement is covered by a transaction, it is guaranteed that concurrent accesses to the records will not harm data consistency. Let T_{move} be the move transaction stated above, T_{old} any older, concurrently running transaction, and T_{new} any newer, concurrently running transaction. T_{old} can read all records deleted by T_{move} in the old partition until it commits and the records are finally removed by GC. T_{old} will not see the records newly created in the target partition, as the creation timestamp/version of the record is higher than its own. T_{new} will also read the records in the source partition, as the deletion version info of those records is older, but refers to a concurrently running transaction. T_{new} will not read the records in the target partition, as they were created by a concurrently running transaction. Any transactions starting after the commit of T_{move} will only see the records in the target partition. These properties follow directly from the use of MVCC.

Using traditional MVCC, writers still need to synchronize access to avoid blind overwrites. For the move transaction, we know that it will not alter the data. Therefore, blindly overwriting the newly created version in the target partition would be acceptable. We have modified the MVCC algorithm in WattDB to allow an exception from the traditional MVCC approach: Records that were moved to another partition can be immediately overwritten, i. e., they have a new version, without waiting for the move to commit.

Cost of reorganization In the following experiments, data is migrated in order to shutdown nodes, thus, reducing the power consumption of the cluster, or data is distributed in order to reduce query response times, which, in turn, also reduces the queries' energy consumption. Moving data is an expensive task, in terms of energy consumption and performance impact on concurrently running queries. We have observed data transfer rates of ~80 Mbit/s in parallel to the OLTP/OLAP workload, hence, it takes less than 2 minutes to move 1 GByte of data from one node to another. The overhead of the move operations 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 with respect to the impact of running queries, the energy cost can be estimated with the duration of the move operation and the (additional) power consumption. Hence, moving 1 GByte of data to a dedicated node with 25 Watts power consumption will require approximately 2.600 Joules.

Monitoring & Control The previously described techniques allow WattDB to dynamically adjust the size of partitions on each node by moving records among them. Thus, we can control the utilization of each of the nodes.

Figure 4 sketches the feedback control loop in WattDB that monitors the cluster, processes the measurements, and takes actions to keep up performance at minimal energy cost. First, the CPU utilization, the buffer hit rate, and the disk utilization on each node is measured and sent to the master node. On the master node, the measurements are evaluated, i. e., each performance indicator is compared to a predefined high and low watermark. If the master detects a

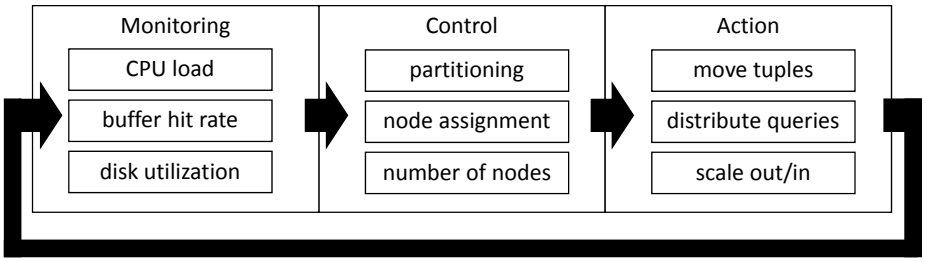


Fig. 4: Monitoring & Controlling the Cluster

workload change and a resulting imbalance in performance or energy consumption, the cluster configuration is examined and actions are evaluated to resolve the issue. Adjustment steps include the re-distribution of partitions among nodes to reduce disk utilization, re-distributing query plans to lower the CPU utilization of nodes, and powering up/down nodes in the cluster to adjust the number of available nodes. For example, the high watermark for CPU utilization is at 85%, hence, exceeding that value will induce the need for another node to help processing queries. Likewise, a disk utilization below 20 IOPS will mark this disk as underutilized and eligible for shutdown. The master node sends out change requests to the cluster nodes, which execute the desired configuration changes, e. g., re-partitioning transactions are started and nodes are powered up and down. Because the master controls transaction execution and query processing, it can rewrite query plans of incoming queries to execute operators on designated, underutilized nodes.

The *master node* is also handling incoming queries and coordinating the cluster. On the master node, a dedicated component, called *EnergyController*, monitors and controls the cluster's energy consumption. This component monitors the performance of all nodes in the cluster. Depending on the current query workload and node utilization, the *EnergyController* activates and suspends nodes to guarantee a sufficiently high node utilization depending on the workload demand. Suspended nodes do only consume a fraction of the idle power, but can be brought back online in a matter of seconds. It also modifies query plans to dynamically distribute the current workload on all running nodes thereby achieving balanced utilization of the active processing nodes.

3 Experimental Setup

The 10 nodes running WattDB and the Ethernet switch are connected to a measurement box which logs the power consumption of each device. A more detailed description of the measurement box can be found in [6]. To submit workloads to the master node, a dedicated DB-client machine, running a predefined benchmark, is used. That machine also aggregates the power measurements and the benchmark results, e. g., throughput and response time, to file. Therefore, we

are able to identify the energy consumption of each benchmark run, even of each query.

In this paper, we are using an adapted TPC-H dataset[6] for the cluster. Some data types of the TPC-H specification are yet unsupported by WattDB and therefore replaced by equivalent types. For example, the *DATE* type was replaced by an *INTEGER*, storing the date as YYYYMMDD, which is functionally identical. Key constraints were not enforced because of the same reason. We pre-created the TPC-H dataset with a scale factor of 100; hence, the DB holds 100 GB of raw data initially. The data distribution strongly depends on the partitioning scheme and the number of nodes online, e. g., with 10 nodes, each node would store ∼10 GB of data. Therefore, the amount of data shipped among nodes under reorganization typically ranges from 100 MB to a few GB.

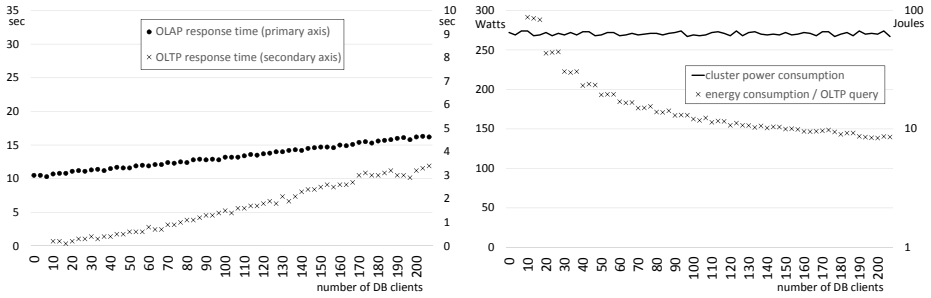
The OLAP part of the benchmark is running TPC-H-like queries that access most of the records in a single query. These queries heavily rely on grouping and aggregation and are therefore memory intensive compared to OLTP queries. TPC-H is a decision support benchmark, hence, we were able to use its queries as analytical workloads. OLAP clients will select one query at a time to run from a list of queries by round-robin. For OLTP, we have taken queries from the TPC-H data generator. In addition, we created corresponding DELETE and UPDATE queries, because the generator is only using INSERT for creating the dataset. Typical OLTP queries are adding/updating/deleting customers and warehouse items; furthermore, they are submitting and updating orders.

A *workload* consists of a single DB client submitting one OLAP query per minute and a given number of DB clients sending OLTP queries. OLTP clients will wait for the query to finish, sleep for 3 seconds of "think time" and start over by submitting a new query. Every 120 seconds, a differing workload is initiated where the number of DB clients may change. Thus, WattDB will have to adjust its configuration to satisfy the changing workloads while keeping energy consumption low. Alltogether, a single *benchmark run* consists of 63 workloads, resulting in a total duration of ∼2 hours.

4 Experimental Results

We have executed four different benchmarks on the cluster. First, we used the benchmark $BENCH_1$ which spawns an increasing number of DB clients sending queries to a fixed 10-node cluster without dynamic reconfiguration. The DB data was uniformly distributed to the disks of all nodes. This experiment serves as the baseline for all future measurements. Next, we ran the same benchmark against a fully dynamic cluster, where WattDB will adjust itself to fit the number of nodes and data distribution to the current workload ($BENCH_2$). Hence, initially all DB data was allocated to the disks of the master node. With growing number of DB clients, the dynamic partitioning scheme initiated a redistribution of the

[6] `http://www.tpc.org/tpch/`



(a) Response time **(b)** Power/energy consumption

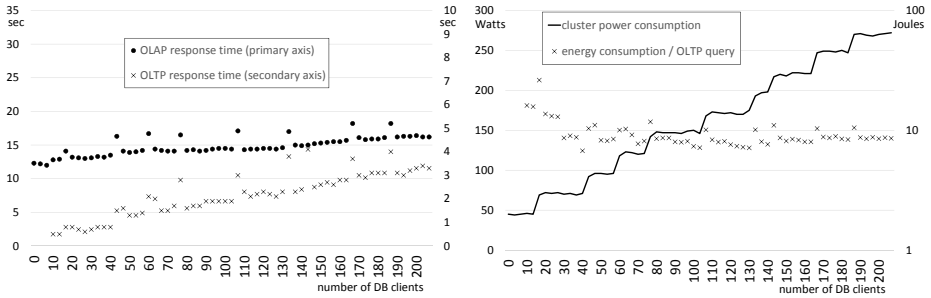
Fig. 5: $BENCH_1$: Increasing load on a static cluster

DB data with each additional node activated[7], such that the data was uniformly allocated to all disks of all nodes at the end of $BENCH_2$. In the third experiment, we shuffled the workload intensities by growing and shrinking the number of (OLTP) DB clients to provide a more realistic, variable workload and, in turn, to provoke more sophisticated partitioning patterns. The cluster was able to react based on the current utilization only, as it did not have any knowledge of upcoming workloads ($BENCH_3$). Finally, we re-ran the benchmark from our third experiment but provided forecasting data to the cluster. Thus, in the last experiment, the cluster could use that information to pre-configure itself for upcoming workloads ($BENCH_4$).

Results of $BENCH_1$: Figure 5a plots the performance of a static database cluster with 10 nodes. All nodes were constantly online and data was uniformly distributed on them. The number of DB clients is increasing over time (x-axis from left to right). Every client is sending OLTP queries sequentially, thus, the number of DB clients defines the number of parallel queries WattDB has to handle. With no OLTP queries in parallel, the cluster takes about 10.5 seconds for an OLAP query. With rising workload, i. e., more parallel queries, response times for OLTP and OLAP queries increase. With 200 clients, the OLAP queries take 16 seconds to finish while OLTP response times increased from 0.2 to 3.3 seconds.[8] Figure 5a depicts the response times for both query types. While the performance of the static cluster is unmatched in all other experiments, its power consumption is the highest. Figure 5b shows the power consumption of the cluster (primary y-axis in Watts) and the energy consumption per query (secondary y-axis in Joules). Obviously, a static configuration will yield higher performance at the price of worse energy efficiency, especially at low utilization levels. As the plots indicate, the energy consumption per query is very high at low utilization.

[7] As far as repartitioning overhead is concerned, frequency and volume of data movement in $BENCH_2$ can be considered as a kind of worst case.

[8] All reported query response times are averages over 120 seconds.



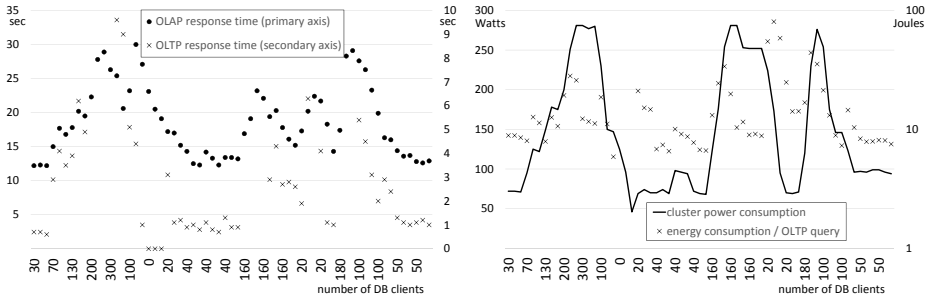
(a) Response time **(b)** Power/energy consumption

Fig. 6: $BENCH_2$: Increasing load on a dynamic cluster

Results of $BENCH_2$: Next, in order to test dynamic reconfiguration ability of WattDB, we have re-run the same benchmark on a dynamic cluster. Figure 6a depicts the performance while increasing the number of DB clients. 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 consumption, as plotted in Figure 6b, is initially low (about 45 Watts for the whole cluster). By calculating the Joule integral over the differing courses of energy consumption in Figures 5b and 6b, one can an impression of the absolute energy saving possible, which is obviously substantial in this experiment.

With increasing utilization, WattDB dynamically wakes up nodes and assigns database 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, average 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 6a visualize the degraded performance while moving data partitions. Likewise, the energy consumption per query increases for a short period of time. Still, no query is halted completely, higher runtimes are a result of increased disk/CPU utilization and wait times because of locked records.

This experiment demonstrates that WattDB is able to dynamically adjust its configuration to varying needs. While minimizing the energy consumption at low utilization, the master node re-actively switches on more nodes as the workload rises. As a result, the energy efficiency of the dynamic cluster is better than the static 10 node configuration, especially at low load levels. Query response times are not as predictable as in a static cluster, because the dynamic reconfiguration places an additional burden on the nodes. Still, the experiment shows that it is possible to trade performance for energy savings.



(a) Response time **(b)** Power/energy consumption

Fig. 7: $BENCH_3$: Varying load on a dynamic cluster

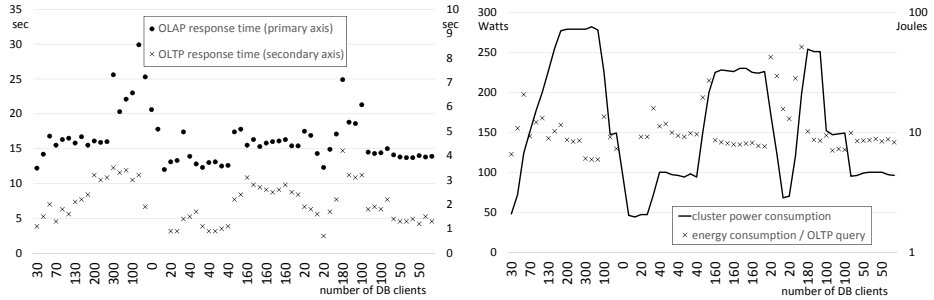
Results of $BENCH_3$: The previous experiments spawned and increasing number of DB clients to submit queries, providing a steadily increasing utilization of the cluster. Thus, the workload was rising slightly over time. Realistic benchmarks require more variance in load changes, in addition with quickly rising and falling demands. Hence, the next experiment employs a more complex pattern of utilization.

In Figure 7a, the x-axis exhibits the number of DB clients over time. This experiment starts with a moderate utilization of 30 parallel clients, climbs up to 300, then quickly drops to idle. Afterwards, two more cycles are starting between low and high utilization levels. This figure plots the performance for both OLTP and OLAP queries, while the cluster is adjusting to the changing workloads. As the graphs indicate, WattDB is heavily reconfiguring and, thus, query response times vary a lot, especially when workloads are shifting.

Figure 7b illustrates the power consumption of the cluster characterized also by high fluctuations as nodes come online and go offline again. WattDB is reacting to changes in the workloads based on its measurements of the nodes' utilization. Therefore, reconfiguration happens re-actively to the specific workload changes. In this experiment with dynamic and extreme workload changes, the reaction time of WattDB is too high to maintain proper query response times. Consequently, energy consumption is also high, mainly due to the overhead of cluster reconfiguration.

This experiment shows that WattDB is able to react to quickly changing workloads, but with less satisfying results. As reconfiguration takes time and consumes resources, a purely reactive adaptation to workloads is not sufficient.

Results of $BENCH_4$: To overcome the limitations of a purely reactive database cluster, WattDB should have some knowledge of the "future" in order to appropriately pre-configure the cluster for upcoming workloads. To test this hypothesis, we have run the same experiment as before (see$BENCH_3$), while WattDB was continuously informed about the following three workloads, i. e., the number of DB clients in the next six minutes. This information can be used



(a) Response time **(b)** Power/energy consumption

Fig. 8: $BENCH_4$: Varying load on a dynamic cluster supported by forecasting

by the master node to proactively partition data to match the worst-case demands of the current and the expected future workload. Figure 8a shows the results for this experiment. The workload on the x-axis was unchanged, but the query response times are more stable compared to those shown in Figure 7a.

On one hand, power and energy consumption, depicted in Figure 8b, are higher under low utilization levels than on a purely reactive cluster (Figure 7b), because WattDB powers up nodes in advance. On the other hand, average energy consumption per query is more predictable, as the query runtimes contain less variance. Furthermore, energy consumption at high utilization levels and at workload shifts is lower, compared to the cluster operated without forecasting data.

5 Conclusion and Outlook

While we already explored opportunities for energy proportionality in the storage layer in [12] and focused on energy-proportional query execution in [13], we have substantially refined and combined both approaches in this research work. In this contribution, we have demonstrated that it is possible to deploy a database system on a dynamic cluster of wimpy nodes. We have exemplified that we can trade energy consumption for query performance and vice versa by controlling the amount of data distribution and number of nodes available to process incoming queries. At the same time, we exhibited that a dynamic cluster is more energy efficient than a statically configured one, where nodes are underutilized—particularly at low load levels.

By keeping data in logical units, partitioned by primary key, WattDB is well suited for OLTP queries, where records are typically accessed by key. The dynamic partitioning enables quick and coherent re-distribution of the data without interrupting data access or putting high overhead on look-up structures. At the same time, the massive parallelism of the cluster nodes running OLTP queries does not interfere with concurrent OLAP queries, where typically large parts of the dataset have to be scanned to calculate aggregates over groups of records.

Moving data among nodes is a time-consuming task and cannot be done very frequently, i. e., on a per-second base. As our experiments indicate, it is crucial for query response times to proactively adjust the cluster to the anticipated workload. Fortunately, workloads typically follow often an easy-to-predict pattern, e.g., workdays are similar to each other, workloads in December keep rising for e-commerce back-end DBMSs, and so on. Therefore, we will focus on forecasting workloads in future experiments.

References

1. Barroso, L.A., Hölzle, U.: The case for energy-proportional computing. IEEE Computer 40(12), 33–37 (2007)
2. Barroso, L.A., Hölzle, U.: The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines. Morgan & Claypool Publishers (2009)
3. Bernstein, P.A., Goodman, N.: Multiversion concurrency control—theory and algorithms. ACM Trans. Database Syst. 8(4), 465–483 (Dec 1983), http://doi.acm.org/10.1145/319996.319998
4. Elmagarmid, A.K.: A survey of distributed deadlock detection algorithms. SIGMOD Rec. 15(3), 37–45 (Sep 1986), http://doi.acm.org/10.1145/15833.15837
5. Härder, T., Reuter, A.: Principles of transaction-oriented database recovery. ACM Computing Surveys 15(4), 287–317 (Dec 1983)
6. Hudlet, V., Schall, D.: Measuring energy consumption of a database cluster. In: BTW, LNI 180. pp. 734–737 (2011)
7. Jung, H., Shim, H., et al.: LRU-WSR: Integration of LRU and Writes Sequence Reordering for Flash Memory. Trans. on Cons. Electr. 54(3), 1215–1223 (2008)
8. Lang, W., Harizopoulos, S., Patel, J.M., Shah, M.A., Tsirogiannis, D.: Towards energy-efficient database cluster design. PVLDB 5(11), 1684–1695 (2012), http://dl.acm.org/citation.cfm?id=2350229.2350280
9. Ou, Y., Härder, T., Jin, P.: Cfdc: A flash-aware replacement policy for database buffer management. In: SIGMOD Workshops, DaMoN. pp. 15–20 (2009)
10. Ou, Y., Härder, T., Schall, D.: Performance and Power Evaluation of Flash-Aware Buffer Algorithms. In: DEXA, LNCS 6261. pp. 183–197 (2010)
11. Park, S., Jung, D., et al.: CFLRU: a Replacement Algorithm for Flash Memory. In: CASES. pp. 234–241 (2006)
12. Schall, D., Härder, T.: towards an energy-proportional storage system using a cluster of wimpy nodes. In: BTW, LNI 214. pp. 311–325 (2013)
13. Schall, D., Härder, T.: Energy-proportional query execution using a cluster of wimpy nodes. In: SIGMOD Workshops, DaMoN. pp. 1:1–1:6 (2013), http://doi.acm.org/10.1145/2485278.2485279
14. Schall, D., Höfner, V., Kern, M.: Towards an enhanced benchmark advocating energy-efficient systems. In: TPCTC, LNCS 7144. pp. 31–45 (2012)
15. Seo, D., Shin, D.: Recently-evicted-first Buffer Replacement Policy for Flash Storage Devices. Trans. on Cons. Electr. 54(3), 1228–1235 (2008)
16. Szalay, A.S., Bell, G.C., Huang, H.H., Terzis, A., White, A.: Low-Power Amdahl-Balanced Blades for Data-Intensive Computing. SIGOPS Oper. Syst. Rev. 44(1), 71–75 (2010)
17. Tsirogiannis, D., Harizopoulos, S., Shah, M.A.: Analyzing the energy efficiency of a database server. In: SIGMOD Conference. pp. 231–242 (2010)