Visualizing the Behavior of an Elastic, Energy-Efficient Database Cluster

Sandy Ganza, Thomas Psota, Daniel Schall, Theo Härder
{s_ganza, t_psota, schall, haerder}@cs.uni-kl.de
Databases and Information Systems Group
University of Kaiserslautern, Germany

Abstract: Energy efficiency in databases is an emerging topic. Our research prototype *WattDB* dynamically adjusts the number of active servers in a cluster to the current workload to achieve energy proportionality. In this demo, we give insights in the partitioning process and *WattDB's* reaction to workload changes by live-presenting a monitoring GUI. The whole process and the resulting configuration are visualized to give immediate feedback, how the cluster would react.

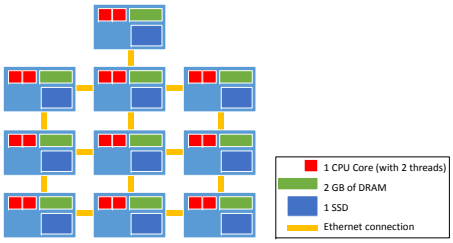
1 Introduction

Today's server hardware is not energy proportional; at low utilization, hardware—mainly main memory and storage drives—consumes a significant amount of power. Hence, about half of the maximum server power is already going to waste when idle. Scaling systems automatically down when idle, thus preventing high idle power consumption, is the main focus of today's servers to provide better energy efficiency. However, this is not enough to approximate energy proportionality.

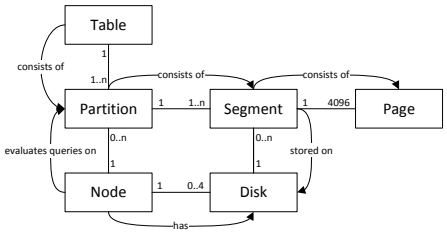
With cloud computing, elastic systems have emerged that adapt their size to the current workload. While stateless or lightweight systems can easily increase or reduce the number of active computing nodes in a cluster, a database faces much more challenges due to the need of high interactions among the nodes and fast reachability of all DB data.

Similar to cloud-based solutions, we hypothesize that a cluster of nodes may adjust the number of active (power-consuming) nodes to the current demand and, thus, approximate energy proportionality. Based on these observations, we developed WattDB, a research prototype of a distributed DBMS cluster, running on lightweight, Amdahl-balanced nodes using commodity hardware. The cluster is intended to dynamically shrink and expand its size, dependent on the workload. Reconfiguring a cluster to dynamically match the workload requires data to be moved from node to node to balance utilization. Yet, copying data is time-consuming and adds overhead to the already loaded cluster. But reducing both, time and overhead, is crucial for an elastic DBMS.

In this demo, we present our recent and ongoing work with WattDB. To give a better understanding of the internal process of balancing and decision making, we show some insights of our DBMS.



(a) The 10-node cluster



(b) Database schema

2 The WattDB Approach

Traditional clustered DBMSs do not dynamically adjust their size (in terms of the number of active nodes) to their workload. Hence, scale-out to additional nodes is typically supported, whereas the opposite functionality, shrinking the cluster and centralizing the processing—the so-called scale-in—, is not. Recently, with the emergence of clouds, a change of thinking occurred and dynamic solutions became a research topic. To test our hypothesis of an elastic, energy-proportional DBMS, that dynamically scales out and back in and that supports SQL query processing with ACID properties, we developed *WattDB*.

The smallest configuration of WattDB is a single server, hosting all database functions and acting as endpoint to DB clients. This server is called *master node*. DB objects (tables, partitions) and query evaluation can be offloaded to arbitrary nodes in the cluster to relieve overloaded nodes, but the master will always act as the coordinator and client endpoint.

To run queries on a cluster of nodes, distributed query plans are generated at the master node. Except data access operators which need local access to the database's records, all query operators can be placed on remote nodes. Running query operators on a single node does not involve network communication among query operators, because all records are transferred via main memory. In contrast, distributing operators implies shipping of records among nodes and, hence, introduces network latencies. Additionally, the bandwidth of the Gigabit Ethernet, which we are using for our experiments, is relatively small, compared to memory bandwidth.

The master node is coordinating the whole cluster. It is globally optimizing the query plans, whereas regular nodes can locally optimize their part of the plan. Furthermore, it takes nodes on- and offline and decides when and how DB tables are (re)partitioned.

Every node is monitoring its utilization: CPU, memory use, network I/O, and disk utilization (storage and IOPS). Additionally, performance-critical data is collected for each DB partition, i. e., CPU cycles, buffer page requests, and network I/O. With these figures, we can correlate the observed utilization of cluster components to (logical) DB entities.

The master checks the incoming performance data to predefined thresholds—with both upper and lower bounds. If an overloaded component is detected, it will decide where to distribute data and whether to power on additional nodes and resume their cluster participation. Similar, underutilized nodes trigger a scale-in protocol, i. e., the master will distribute the data (processing) to fewer nodes and shutdown the nodes currently not needed.

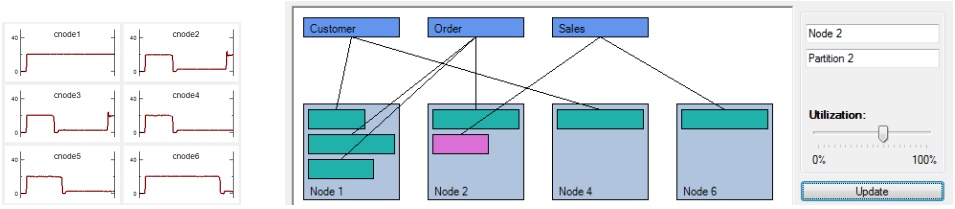
In WattDB, we provide different policies regarding the scale-out behavior. First, each node in the cluster stores data on local disks to minimize network communication. If storage space of a node is in short supply, DB partitions are split up on nodes with free space. In this way, it tries to keep the I/O rate for each storage disk in a certain range. Underutilized disks are eligible for additional data—either newly generated by INSERT operations or migrated from overloaded disks. Utilization among storage disks is first locally balanced on each node, before an allocation of data from/to other nodes is considered. Third, each node's CPU utilization should not exceed the upper bound of the specified threshold (80%) for more than short utilization spikes. As soon as this bound is violated on a node for longer than one minute, WattDB first tries to offload query processing to underutilized nodes.[1] If the overload situation cannot be resolved by redistributing the query load, the current data partitions and their node assignments are reconsidered. When a partition causing the overload is identified, it is split according to the partitioning scheme applied, where affected segments are moved to other nodes [SH15]. For underutilized nodes, an inverse approach is needed. A scale-in protocol is initiated, which quiesces the involved nodes from query processing and shifts their data partitions to nodes currently having sufficient processing capacity.

Similar rules exist for network and memory utilization, e. g., if the working sets of the transactions become too big for the DB buffer, repartitioning is triggered. The master makes decisions based on the current workload, the course of utilization in the recent past, and the expected future workloads [KHH12]. Additionally, workload shifts can be user-defined to inform the cluster of an expected change in utilization.

WattDB utilizes a rather simplistic DB schema with logically partitioned tables. A *table* is a purely logical construct in WattDB. Its metadata (column definitions, partitioning scheme) is maintained 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). Each partition contains 1 to m *segments* (of 32 MB), which are physical units of storage. Each segment is located on a local disk to reduce access latencies. Partitions are by default index-organized [Sea00] w. r. t. the primary key and can be supported by additional, secondary indexes. Fig. 1b clarifies these terms and their relationships.

To facilitate speedy rebalancing, we have implemented *physiological partitioning* [SH15] to be able to quickly repartition the cluster in case of workload imbalances. With physiological partitioning, each segment is a self-contained subpartition, including a defined primary-key range. The partition's metadata only stores a list of segments with their respective primary-key ranges. Therefore, segments can be detached from one partition and copied to another node, with minimal impact to query processing. Since segments are self-contained, no outside references need to be changed during rebalancing, except updating metadata in the partition and on the master. We have experimentally verified the advantages of physiological partitioning vs. other approaches and conclude that our system is well-suited for an elastic DB cluster. Yet, details about the repartitioning process inside the master node cannot be discussed here [SH15]. In this demo, we exhibit how physiological partitioning allows the master to quickly rebalance the cluster.

[1] This works well for operators like **SORT**, **GROUP**, and **AGGREGATE**.



(a) Power consumption

(b) Partitioning and utilization

Figure 2: Graphical user interface

3 Demo Description

We are going to present how partitioning in WattDB works and how workload changes trigger rebalancing of DB partitions. We put up a GUI, showing the live status of the cluster (via remote connection). Demo visitors will see the hardware state of the nodes, i. e., whether they are online or offline, and their corresponding power consumption. Additionally, a predefined DB schema with its tables and partitions is sketched. The assignment of partitions to nodes and the current utilization of nodes and partitions is also visible. Hence, the GUI will show both, the underlying hardware as well as the DB layout on top.

Users will be able to alter the workloads by increasing or reducing utilization of tables and partitions. The new workload profile is sent to WattDB, where the master node will then trigger rebalancing. The audience can see, how the cluster rebalances, moves data, and potentially splits and merges partitions to adjust to the modified workload. The master node will power up additional nodes, if necessary, which will affect overall power consumption. Likewise, superfluous nodes are shut down and the outcome is visible on the GUI.

Since rebalancing data is time-consuming and we do not expect visitors to wait several minutes, the master nodes does not send out actual rebalancing commands to the nodes. Instead, only the process on the master node is depicted, showing how WattDB's decision process works and what partitions are deemed candidates for rebalancing. Hence, the audience will be able to see the reaction of the cluster almost instantaneously.

References

[KHH12] Christopher Kramer, Volker Höfner, and Theo Härder. Lastprognose für energieeffiziente verteilte DBMS. *Proc. 42. GI-Jahrestagung 2012, LNI 208*, pages 397–411, 2012.

[Sea00] Jagannathan Srinivasan and et al. Oracle8I Index-Organized Table and Its Application to New Domains. In *Proc. VLDB Conf.*, pages 285–296, 2000.

[SH15] Daniel Schall and Theo Härder. Dynamic Physiological Partitioning on a Shared-nothing Database Cluster. In *ICDE Conf.*, 2015.