Energy Efficiency in Database Systems

Vom Fachbereich Informatik
der Technischen Universität Kaiserslautern
zur Verleihung des akademischen Grades

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation

von

Dipl.-Inf. Daniel Schall

Dekan des Fachbereichs Informatik:

Prof. Dr. Klaus Schneider

Prüfungskommission:

Vorsitzender:	Prof. Dr. Christoph Grimm
Berichterstatter:	Prof. Dr.-Ing. Dr. h. c. Theo Härder
	Prof. Dr. Jens Teubner

Datum der wissenschaftlichen Aussprache:

26. Februar 2015

For Peggy and Maja-Sophie

Acknowledgements

In the past five years, I spent as a scientific staff member and PhD candidate in the Database and Information Systems research group at the University of Kaiserslautern, I met a lot of valuable people. Without their support, encouragement and friendship, my work would not have been possible and I am very thankful to everyone I became acquainted with along my way.

First, I want to express my deepest gratitude to my advisor, Prof. Dr.-Ing. Dr. h. c. Theo Härder for his offer to work with him and his constant support, while still giving me the freedom to research on my own. He always had an ear for me when I needed some helpful advice and his guidance and encouragement has led this thesis to success. Without his passionate lectures about database fundamentals, which jump-started my fascination with the topic, and his offer to join his team, I would not have taken *the road less traveled by* [Fro16] and started my carreer in research.

Next, I would like to thank Prof. Dr. Jens Teubner for dedicating a considerable amount of time and effort into reviewing this work and providing lots of insightful comments. I had the pleasure of meeting Prof. Teubner on various conferences, and he accepted the role of second examiner without hesitation.

I would also like to thank the head of the examination board, Prof. Dr. Christoph Grimm, for providing a pleasant atmosphere during the defense of my thesis and for his inspiring remarks.

My acknowledgements go out to my studies advisor, Prof. Dr.-Ing. Stefan Deßloch, who supported and counseled me during my computer science studies. He also taught me many fundamentals of information systems and was always available for valuable discussions.

"Standing on the shoulders of giants" is a phrase often used when referring to fundamental and related work. Yet, not only the giants we are standing on are worth mentioning, but even more the ones we encounter every day. During my time as a researcher, I had the pleasure to work and discuss with many *giants*, sharing ideas, offering help and giving priceless advice. Withouth them, I would never have come that far. Therefore, I am very grateful to having (had) such wonderful colleagues and friends, especially Dr. Karsten Schmidt, Dr. Sebastian Bächle, Dr. Andreas M. Weiner, Dr. Yi Ou, Dr. Thomas Jörg, Dr.

Jürgen Göres, Dr. Boris Stumm, Dr. Christian Mathis, Dr. Philipp Dopichaj, Andreas Bühmann and Joachim Klein. With them, I shared an incredible amount of time during the last ten years, and they brought me both, a lot of experience and even more fun each day. It was a great pleasure working with them.

Yet, with one of my former colleagues, I shared more than just time, we also shared an office and our vision of an energy-efficient DBMS. I am deeply grateful to Volker Höfner for his advice, his dedication to our project, and, foremost, for our friendship we shared from the first day we started studying in Kaiserslautern. His work on the foundations of our DBMS paved the path for many publications and ultimately, this thesis.

Further, I would like to thank all my former research fellows, with whom I had a great time in Kaiserslautern. With Caetano Sauer, Johannes Schildgen, Weiping Qu, Yong Hu, and Martin Grössl it was always a pleasure to work and we had many interesting discussions, not limited to research-related issues.

Also, I would like to thank our numerous research assistants that worked on our project and helped implement WattDB: Thomas Psota, Juri Krieger, Sandy Ganza, Vitor Uwe Reus, Pedro Martins Dusso, Lucas Dos Santos Lersch, Don Kuzhiyelil, Christopher Kramer, and Lars Scherer. By dedicating their time and skills, they heavily supported my project.

Last but not least, I want to thank my family for their love and support. From my first steps with computers on, my parents Beate and Gerhard, and my brother Tobias always supported me in pursuing my carreer in computer science, and discovered many, many ways to challenge my skills in computer repairs. Also, I would like to express my deepest gratitude to my wonderful wife Peggy, who constantly encouraged me to pursue my dreams, always had patience and gave love unconditionally, through good and bad times. Without her wind beneath my wings [Mid89], this thesis would not have been possible. Finally, I want to thank my daughter Maja for reminding me of a world outside my office. Her smile and laugh often helped me through a rough patch.

Daniel Schall

Walldorf, March 2015

Abstract

Today, database management systems are highly complex, ubiquitous pieces of software. Traditionally, the main focus in DBMS development was perfomance-centric, hence, heavy-weight systems with peak performance were developed.

In the last years, electricity prices have increased significantly, and hence, the motivation to save power came into focus of datacenters and into the DBMS community. Recent ventures on increasing energy efficiency of databases did contribute only marginal improvements.

In this thesis, we present our findings on an elastic cluster of lightweight nodes, that is able to dynamically adapt to the workload by scaling out and back in. By powering down unused nodes, we avoid high idle power consumption of today's hardware. Our contribution can be summarized as follows:

- We present a newly developed benchmarking paradigm for evaluating energy-efficiency, since existing DB benchmarks do not include the whole power spectrum in their measurements.
- We describe our work on measuring and benchmarking energy efficiency of commodity hardware with a custom-made measurement track, capable of metering power and energy consumption of an entire server cluster and correlating the readings with benchmark results.
- We studied and improved our theories on our own database management system, called WattDB, which we also present in detail in this thesis. WattDB is a distributed DBMS, running on lightweight, commodity hardware, with energy efficiency as prime optimization goal. Since the database is distributed among all nodes in the cluster, we also introduce techniques necessary for repartitioning to keep data available at all times. Finally, we compare our approach to a traditional server in terms of power and performance.

Contents

Abstract vii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Objective . . . 2
- 1.3 Overview . . . 3
- 1.4 Disclaimer . . . 3

2 Related Work 5
- 2.1 Energy Efficiency . . . 5
  - 2.1.1 Components . . . 5
  - 2.1.2 Single machine . . . 8
  - 2.1.3 Amdahl-balanced components . . . 20
- 2.2 Distributed Systems . . . 21
  - 2.2.1 Distributed and energy-efficient storage . . . 23
  - 2.2.2 Dynamic clustering . . . 24
  - 2.2.3 Database partitioning . . . 25
  - 2.2.4 Partitioning examined . . . 27

3 Measuring Energy Consumption 33
- 3.1 Metrological Background . . . 33
  - 3.1.1 Charge . . . 33
  - 3.1.2 Voltage . . . 33
  - 3.1.3 Current . . . 34
  - 3.1.4 Alternating current . . . 34
  - 3.1.5 Resistance . . . 34
  - 3.1.6 Water circuit analogy . . . 35
  - 3.1.7 Power . . . 36
  - 3.1.8 Work . . . 36
- 3.2 Consumption of Electricity . . . 37
- 3.3 Cost of Electricity . . . 39

3.4 Energy Efficiency . . . 40
3.5 Energy Proportionality . . . 41
3.6 Measuring Power & Energy Consumption . . . 42
3.6.1 Voltage . . . 43
3.6.2 Current . . . 43
3.6.3 Power . . . 43
3.7 Measurement Device . . . 44
3.7.1 Problem description . . . 44
3.7.2 Existing measurement solutions . . . 45
3.7.3 Single-server measurement device . . . 49
3.7.4 Cluster measurement device . . . 50
3.7.5 Accuracy estimation . . . 52

4 Benchmarking for Energy Efficiency 59
4.1 Related Work . . . 59
4.1.1 TPC-C . . . 59
4.1.2 TPC-E . . . 60
4.1.3 TPC-H . . . 60
4.1.4 TPC-Energy . . . 61
4.1.5 SPECpower_ssj2008 . . . 62
4.1.6 Server efficiency rating tool . . . 63
4.2 JouleSort . . . 64
4.3 Benchmark Requirements . . . 64
4.4 Benchmark Proposal . . . 65
4.4.1 Static weighted energy proportionality . . . 67
4.4.2 Dynamic weighted energy efficiency . . . 71
4.4.3 Energy delay product . . . 73
4.5 Benchmark Software . . . 74

5 The WattDB Framework 77
5.1 Design Principles . . . 77
5.2 Cluster Hardware . . . 77
5.2.1 Power consumption . . . 78
5.2.2 GPU . . . 79
5.2.3 Amdahl-balance . . . 81
5.3 Software Design . . . 81
5.3.1 Master node . . . 82
5.3.2 Communication . . . 82

5.4 Performance Monitoring 83
5.4.1 Physical probes 83
5.4.2 Logical probes 84
5.4.3 Initial implementation 85
5.5 Forecasting 86
5.6 Query Optimizations 88
5.6.1 Operator optimizations 88
5.6.2 Plan optimizations 93

6 Shared Disk vs. Shared Nothing 101
6.1 Traditional Definitions 101
6.2 Shared-* in WattDB 104
6.2.1 Virtual storage network 105
6.2.2 Storage segmentation 105
6.2.3 Conclusion 107

7 Elastic Storage 109
7.1 Elasticity in the Storage Layer 109
7.2 Design Overview 110
7.3 Experimental Evaluation 113
7.3.1 Simulating OLTP queries 113
7.3.2 Initial test run 114
7.3.3 Results 115
7.4 Skewed Access Patterns 118
7.5 Workload Shifts 119
7.6 Optimizations 120
7.6.1 SSDs 120
7.6.2 Page compression 122
7.7 Summary 124

8 Elastic Query Processing 125
8.1 Elasticity in the Query Processing Layer 125
8.1.1 Design overview 125
8.1.2 Distributing queries 126
8.1.3 Experiments 131
8.2 OLTP Workloads 138
8.2.1 Implementation 139
8.2.2 Experiments 140

8.2.3 Results 140
8.3 Summary 143

9 Elastic DBMS **145**
9.1 Architecture 145
9.1.1 Storage structures and indexes 146
9.1.2 Dynamic partitioning scheme 146
9.1.3 Concurrency control 148
9.1.4 Move semantics 150
9.1.5 Cost of reorganization 151
9.1.6 Monitoring & control 151
9.2 Experimental Setup 152
9.3 Experimental Results 153
9.4 Summary 160
9.5 Dynamic Aspects of Partitioning Revisited 160
9.5.1 Physical partitioning 161
9.5.2 Logical partitioning 162
9.5.3 Physiological partitioning 162
9.6 Experiments 165
9.6.1 Experimental setup 166
9.6.2 Results 167
9.7 Summary 174

10 Comparison with a Brawny Server **175**
10.1 Total Cost-of-Ownership Analysis 175
10.1.1 TCO model 175
10.1.2 Hardware selection 176
10.1.3 Workload profile 178
10.1.4 Scale-out policy 178
10.1.5 Results 179
10.2 Experimental Examination 180
10.2.1 Hardware 180
10.3 Experiments 183
10.3.1 Experimental setup 183
10.3.2 Performance-centric benchmark 184
10.3.3 Energy-centric benchmark 186
10.3.4 Dynamic workloads 190
10.4 Summary 197

11 Summary and Future Work **199**
11.1 Conclusion 199
11.2 Outlook 200

A Appendix **203**
A.1 WattDB's Evolution 203
A.2 TPC-H Queries 206
A.2.1 TPC-H Q1 206
A.2.2 TPC-H Q4 207
A.3 Record Server 207

Bibliography **211**

List of Figures

1.1 Worldwide cost to power and cool servers 2

2.1 Clock rate and power for Intel processors 6
2.2 Main-memory power consumption 7
2.3 Power by component at different activity levels 8
2.4 Average CPU utilization 9
2.5 Power consumption and energy efficiency 10
2.6 Server utilization histogram 11
2.7 Server power breakdown 12
2.8 Average CPU and disk utilization of SPH's servers . . . 13
2.9 Hourly CPU utilization of two servers 14
2.10 Performance and energy consumption of a TPC-C trace 16
2.11 Breakdown of average power 18
2.12 Power use over single-node system utilization 20
2.13 Database schema 27
2.14 Physical partitioning 32
2.15 Logical partitioning 32
2.16 Physiological partitioning 32

3.1 Analogy between water and electric circuit 35
3.2 Voltage, current, and real power in AC 37
3.3 US datacenter electricity use 38
3.4 US industry-sector electricity prices 39
3.5 Measuring voltage and current 42
3.6 Measurement track and required peripherals 48
3.7 Plug-in module for measuring power consumption 51
3.8 Labjack overview 51
3.9 Software overview 52
3.10 Setup for accuracy estimation of the Labjack UE9 . . . 53
3.11 Results from Labjack UE9's accuracy estimation 53
3.12 Accuracy estimation of the measurement device 54

3.13 Results from the device's accuracy estimation 55
3.14 Accuracy estimation of the measurement track 56
3.15 Results from the measurement track's accuracy estimation 56
3.16 Long-time accuracy estimation 58

4.1 TPC-Energy Measurement System 62
4.2 SERT design overview 63
4.3 TPC-* vs. the real world 66
4.4 Static energy-efficiency measurement 67
4.5 Dynamic weighted energy-efficiency sample 72
4.6 Architecture of the benchmark software 74

5.1 GPU framework in WattDB 80
5.2 WattDB's monitoring framework 83
5.3 Architecture of WattDB's monitoring 86
5.4 WattDB's monitoring & forecasting 87
5.5 Example operator tree for TPC-H query 4 89
5.6 Iterator model 90
5.7 Vectorization 90
5.8 Vectorized and non-vectorized operator performance . . 91
5.9 Partition pruning 94
5.10 Partition-wise join 96
5.11 Iterator model 97
5.12 Offloading operators: Performance comparison 98

6.1 Classification of distributed DBMS 102
6.2 Shared-disk and shared-nothing architectures compared 103
6.3 WattDB's virtual storage-area network 104
6.4 WattDB's disk segmentation 106

7.1 Decoupled storage 109
7.2 5-layer database architecture 110
7.3 Shared-disk design overview 111
7.4 OLTP trace on a dynamic cluster 116
7.5 Effects of access-pattern variations 118
7.6 OLTP trace on a dynamic SSD-based cluster 121

8.1 Elastic query processing on shared disk 126

8.2 Experimental setup 127
8.3 Energy and performance of distributing queries 129
8.4 Varying query load of TPC-H query Q1 132
8.5 Varying query load of TPC-H query Q4 133
8.6 Experiments on three cluster configurations 137
8.7 Lock hierarchy in WattDB 139
8.8 Distributing TPC-C query operators 141

9.1 Three steps of a split in the partition tree 147
9.2 MVCC vs. locking: Performance and storage 149
9.3 Monitoring & controlling the cluster 151
9.4 Increasing load on a static cluster 155
9.5 Increasing load on a dynamic cluster 156
9.6 Varying load on a dynamic cluster 158
9.7 Varying load on a dynamic, forecasting cluster 159
9.8 Benchmark results for different partitioning schemes . . 167
9.9 Benchmark results for different partitioning schemes . . 168
9.10 Impact factors on query runtime when rebalancing . . . 170
9.11 Improving physiological partitioning 172
9.12 Improving physiological partitioning 173

10.1 The 10-node cluster compared with the brawny server . 180
10.2 Power consumption for both systems 181
10.3 Theoretical performance figures for both systems 182
10.4 Peak performance and energy cons. for both systems . . 185
10.5 Performance and energy consumption for OLAP 187
10.6 Performance and energy consumption for OLTP 189
10.7 Dynamic OLAP workload on both systems 192
10.8 Dynamic OLTP workload on both systems 194
10.9 Overall results for big server and cluster 196
10.10EDP of OLAP & OLTP workloads 197

A.1 Evolution of WattDB, 2009 to 2012 204
A.2 Evolution of WattDB, 2012 to 2015 205
A.3 TPC-H Q1 on a 1-node storage cluster 208
A.4 TPC-H Q1 on a 3-node storage cluster 208
A.5 TPC-H Q1 on a 5-node storage cluster 209
A.6 TPC-H Q1 on a 7-node storage cluster 209

List of Tables

3.1 Points of measurement for a single server 49
3.2 Points of measurement in a cluster 50
3.3 Long-running accuracy estimation measurement 57

4.1 Example calculation of the SWEP 69

5.1 Power consumption per component 79
5.2 Sample round-robin database 85

7.1 Effects of access-pattern variations 119
7.2 Effects of compression on performance 123

8.1 Experimental results from distributing queries 128
8.2 Results from distributing OLTP operators 142

10.1 Hardware components 177
10.2 TCO comparison at 0.20 € / kWh electricity 179

1 Introduction

Traditionally, database systems have been evaluated regarding their performance in terms of throughput and response times and the corresponding cost. Yet, with increasing energy prices and higher energy density in datacenters, leading to problems supplying electricity and removing waste heat efficiently, energy efficiency became another optimization goal. Therefore, techniques to measure, evaluate, and improve energy efficiency have been published in the last years. In this work, we give an overview over our research (and related approaches) on an energy efficient database system.

1.1 Motivation

Power consumption and energy savings are concerns in all areas of IT. Recently, a study revealed that IT—including not only datacenters, but also communication facilities, cellular devices, phones, home entertainment systems, and basically all computerized machines—is responsible for 10% of the world's electricity consumption: 1,500 terawatt hours of energy per year [Mil13].

Datacenters in turn, are responsible for a fraction (∼12%) of the whole IT's power consumption. In 2006, US datacenters and servers consumed 61 billion kWh, costing 4.5 billion USD [LP09]. As of 2007, US datacenters consume about 1.5% of the country's total electricity generation, projected to 3% in the next years [EPA07].

While energy consumption is steadily rising, prices for electricity are also increasing. Due to the increasing power consumption of computers, electricity cost of datacenters is a dominant factor in the total cost of ownership [BH07]. Each watt converted into heat in the servers needs to be cooled off again; hence, cooling expenses draw level with power cost [BH07]. For an average utilization period (∼5 years), energy costs have now drawn level with the server's acquisition cost [PN08].

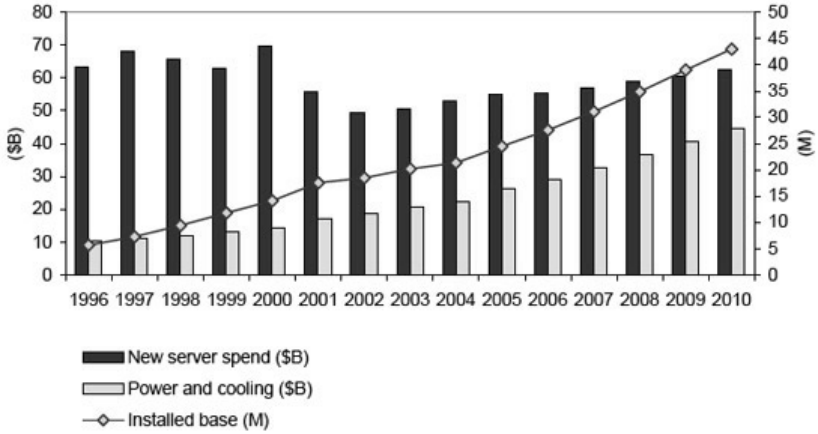


Figure 1.1: Worldwide server cost between 1996 and 2010, from [PN08]

Until recently, energy efficiency was only a hot topic in embedded and mobile environments, but neglected in datacenter and server design. In recent years, several proposals have been published to reduce overall energy consumption of database servers and to improve energy efficiency of such systems. As consequence of such efforts, only marginal improvements on simplified problems were achieved.

1.2 Objective

In order to substanitally improve energy efficiency of database servers, the limitations of non-energy-proportional hardware have to be overcome. While single-server-based approaches have already provided small advances in energy efficiency, at idle, 50% of the peak power consumption is still wasted.

Therefore, we have worked on a distributed solution, implementing an elastic database management system on a cluster of lightweight nodes. By dynamically scaling the size of the cluster to the workload demands, we are able to tune the power consumption and performance on a per server basis.

1.3 Overview

In this work, we present our work on an energy-proportional database management system. First, we give an overview of related work in all nearby fields of research, covered by this thesis. We outline efforts on improving energy efficiency of components and servers, up to distributed systems. Further, we present performance optimization techniques for distributed database systems leading to better energy efficiency.

Afterwards, we summarize the metrological foundations, how to measure power and energy consumption of servers, give an overview on current power measurement devices, and—since no existing solution fits our purpose of monitoring a database cluster—explain the development of our own measurement track in detail. Next, we compare different system benchmarks, both performance-centric and energy-related, and introduce our proposal for an energy-centric benchmarking paradigm.

In Chapter 5, we present our core contribution, *WattDB*, an energy-proportional database system. We describe the system's architecture and design rationale behind it, analyze its power consumption, and outline imporant implementation aspects.

Later, we work through the database layers from storage to processing and outline techniques to gain better energy efficiency on each layer. In Chapter 7, we present our work at the DB's storage layer, work our way up through the processing layer in Chapter 8, and combine our approaches to form a fully elastic database system in Chapter 9. Next, we compare the performance and energy consumption of our implementation to a big server, running the same version of our DBMS.

Finally, we give a conclusion and an outlook on future work.

1.4 Disclaimer

WattDB is an ever-changing research project and has undergone some remarkable evolution from the first implementation for the SIGMOD programming contest [Gen+10]. Starting with very limited query evalutation capabilites, it later supported analytical queries, multi-version concurrency control and ACID-compliant updates.

In this work, we refer to WattDB at various development stages in chronological order. As we proceed in the text, more features were im-

plemented, design principles were reviewed and architectural changes were made. For example, in Chapter 8, we implemented the functionality to process updates and, thus, our DBMS turned from read-only to write-supported. Therefore, some statements about our DBMS seem contradictory when read ouf of context.

We tried to clearly note the current state of development in each chapter, but we also encourage the reader to keep the evolution of WattDB in mind when reading. In Appendix A.1, we give a short chronology of WattDB's evolution as a quick reference.

2 Related Work

Gaining better energy efficiency has been a hot topic lately. Since the research area is still emerging and focus has just recently shifted to improving energy efficiency in database servers, the amount of publications is countable. Yet, efforts on reducing power consumption started very early, although with another focus.

In the following, we give an overview over related work in the fields of energy efficiency in general and efforts in the DBMS community to lower power consumption. More related research specialized to topics covered in this thesis can be found in the respective chapters.

2.1 Energy Efficiency

Improving energy efficiency has been an effort in many areas of IT. In the following, approaches on various "granularities", from chip-level over entire machines up to datacenter-grade solutions, are presented.

2.1.1 Components

At the small scale, microchip manufacturers have to carefully design their chips with respect to power consumption. First, with shrinking die[1] sizes, the area for heat dissipation became smaller and, thus, cooling the chip became more difficult. Second, smaller structures need reduced voltages to function properly.

With the emergence of more powerful processors, running at higher frequencies, the wattage of computer systems steadily increased. In the late 2000s, microchips hit the so-called "power wall" [Kur01], when chip frequencies could no longer increase as before, because they would consume too much electricity and, thus, generate too much heat. Now thermal constraints prevented systems from clocking higher and higher

[1] The area of the chip is called *die* [Aye03, p. 31].

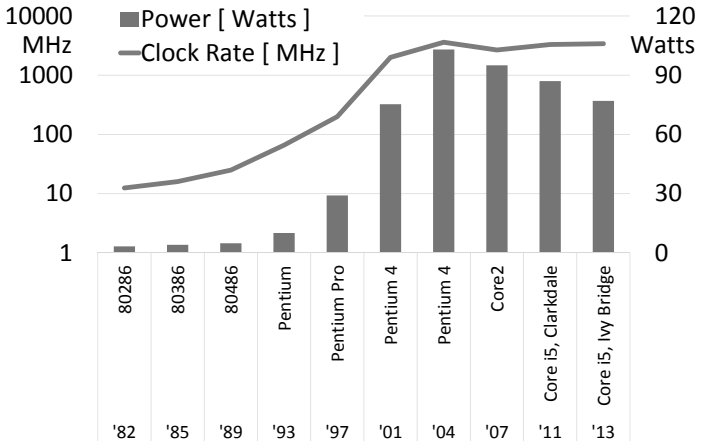


Figure 2.1: Clock rate and power for the last Intel processor generations, from [PH13, p. 40, Fig. 1.16]

as in the years before and the heat generated in the CPUs just could not get cooled down efficiently any more.

In recent CPU generations, improvements have not been made in terms of higher clock rates, as observed before, but in the number of parallel cores on each die, as Figure 2.1 exemplifies. Since power consumption scales quadratically with the supplied voltage ($P = \frac{U^2}{R}$), raising the voltage dramatically increases the power draw of an integrated circuit. Therefore, higher clock rates (which require higher voltages) were no longer desirable. Instead, parallelization and dynamic frequency control came to play, e. g., processors were designed to power down when underutilized [Int04] [Kim+08].

Especially in notebooks, mobile devices and sensors, energy efficiency was a critical property from the very start of these technologies. On the opposite, server and datacenter operators did not care much about reducing power consumption. Power and heat were merely technical constraints that had to be taken into account when planning large datacenters as well as single systems.

As already mentioned in the introduction, views changed with the increase of electricity prices and the growing installation base of servers

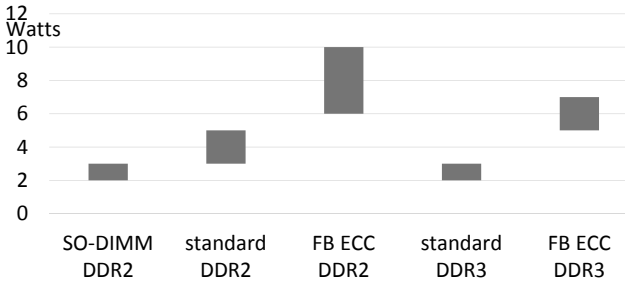


Figure 2.2: Power consumption range of various main memory modules

[EPA07]. Also, trends from portable computers were introduced into server-grade hardware, for example, *Dynamic Voltage and Frequency Scaling*.

Although these techniques greatly improve operation time on battery in mobile devices and notebooks, the effects in server hardware are less dramatic, mainly because of different utilization patterns. End-user devices are used infrequently with long periods of idleness between utilizations. Therefore, suspending components, primarily CPU, hard disk, and LCD screen, has great impact on battery time.

While CPUs have shown improvements in energy efficiency, other computer components did not improve at all in the last decades. Main memory, for example, is a big energy consumer in every system. While there have been improvements in overall energy consumption of main memory, especially for mobile, battery-powered systems, the power consumption of server-grade main memory did not reduce much.

Typical DDR2 SO-DRAM (for notebooks) requires approx. 2 to 3 watts per 1GB module [Mic04]. Comparable DDR2 DRAM (for desktop computers) already consumes 3 to 5 watts for the same size [Zhe12]. Fully-buffered DDR2 ECC RAM (for servers) needs 13 W for a single 2GB module and, thus, requires far more power [Fis12].

While DDR3 RAM modules in general promise lower power consumption than DDR2 modules, they still do not offer dynamic power savings mechanisms and, thus, are still static power consumers [Mic07]. An 8GB DDR3 RDIMM ECC module, for example, still consumes over 6 watts [Fis12]; yet it is an improvement compared to the previous generation.

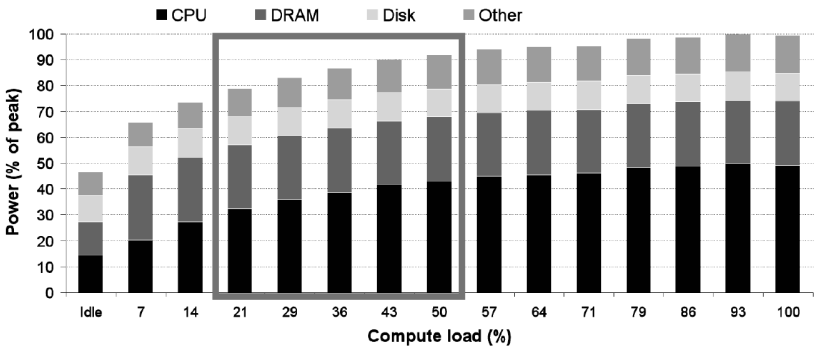


Figure 2.3: Power by component at different activity levels, from [Spe08]

Figure 2.2 depicts the power consumption of different types of main memory. Main-memory modules cannot be selectively turned off or suspended to reduce energy consumption. Therefore, main memory is a big, constant power consumer in today's servers.

The fact, that there is little literature about the power consumption of commodity memory modules, speaks for itself.

Hard disk drives are used to store data persistently. Traditional drives have mechanical parts (rotating magnetic platters, moving arm with read/write heads) that consume electricity. Additionally, the drive's platters are constantly spinning, consuming energy regardless of utilization. As of 2013, disks feature storage sizes of 1 TB per platter and up to 4 platters per device.

With the emergence of SSDs, a new storage technology was competing with traditional HDDs. While SSDs offer higher IOPS and lower access latencies, the price per Gigabyte storage is still about a magnitude higher compared to hard disks. Energy consumption of SSDs largely varies, depending on the model and flash chips used [HS11b].

2.1.2 Single machine

Several studies measured the energy consumption of a single server and analyzed energy efficiency under various workloads. In 2008, Spector showed a typical power breakdown of server components under various

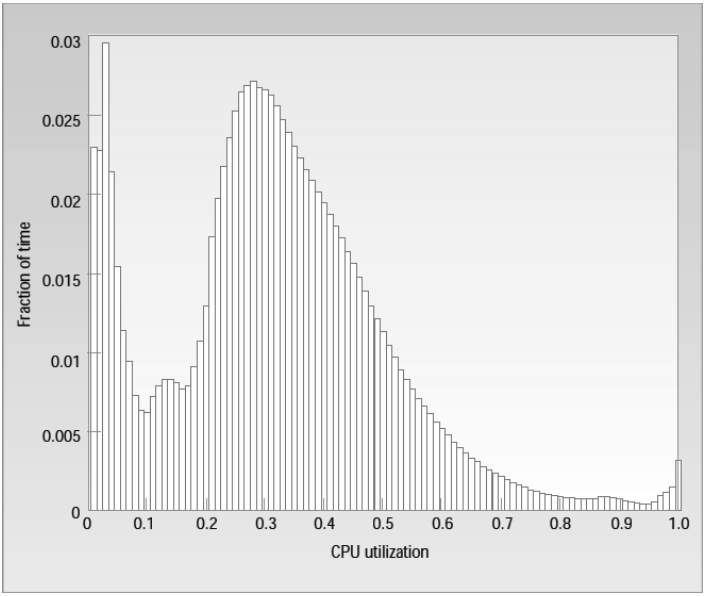


Figure 2.4: Average CPU utilization, from [BH07]

load levels [Spe08], see Figure 2.3. As the graph exhibits, computers consume the most power at full utilization, but still require about 50% of peak power consumption at idle. With increasing workload, power consumption quickly rises to its peak.

Lots of similar studies confirmed the same relationship between utilization and power consumption, varying only in details [HS11a; Lan+12]. As main conclusion, a server consumes about half its peak power at idle and super-linearly increases when utilized.

Based on similar findings, Louis Barroso and Urs Hölzle published a study that shows performance data of Google's MapReduce server cluster [BH07] in 2007. Figure 2.4 charts the aggregate histogram for the CPU utilization of 5,000 servers hosted at Google, while Figure 2.5 plots their power consumption and energy efficiency. According to this study, the servers are typically operating at 10% to 50% of their maximum performance. That way, servers are barely idle but, as well, barely fully utilized.

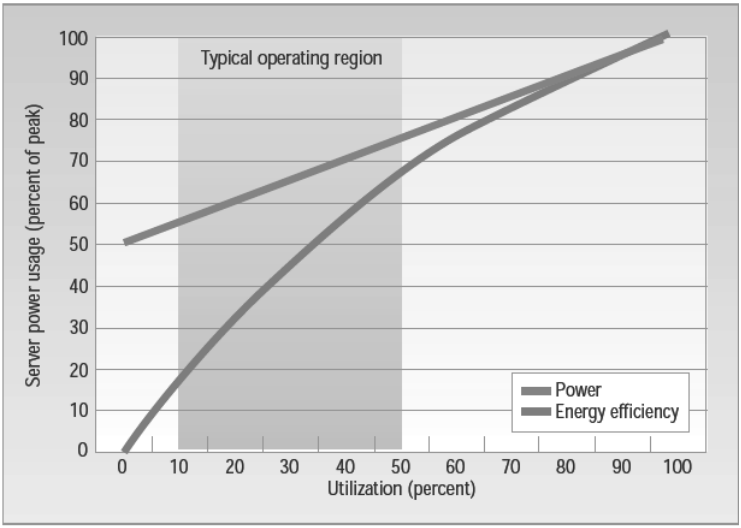


Figure 2.5: Power consumption and energy efficiency, from [BH07]

Meisner, Gold, and Wenisch examined production workloads of 120 enterprise-servers. They identified the CPU and I/O subsystem as the major power consumers, as plotted in Figure 2.7. The servers' average utilization is graphed in Figure 2.6. Similar to the previously presented experiments, these studies indicate that typical workloads do not keep the servers busy for significant periods of time.

In 2011, we conducted a study similar to the one perfomed by Google and monitored SQL and business intelligence (BI) servers of SPH AG [SHK12]. SPH is a mid-sized ERP-developing company that specializes in the branches mail order and direct marketing. Its ERP products are based on IBM System i5 or (in the considered case) on Microsoft Dynamics AX. For some of its customers, SPH AG is hosting the ERP servers in-house, including SQL servers and BI servers. The SQL servers are used to store the ERP data, such as customer, sales order, and invoice information. For the ERP system, 24/7 availability is also needed, because on-line shops are connected to the ERP systems. The BI servers are used to process data of the SQL servers for the preparation of re-

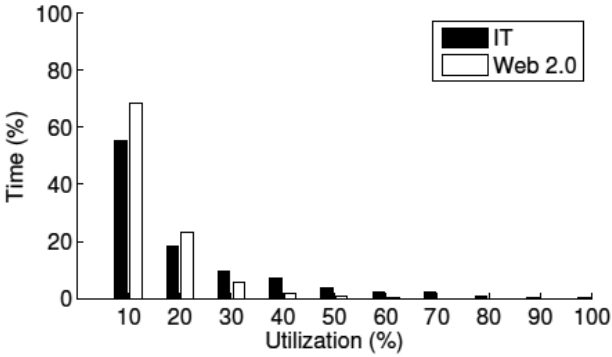


Figure 2.6: Server utilization histogram, from [MGW09]

ports for company management. This data is updated by a nightly job. On all servers, thorough performance and load monitoring is installed.

We analyzed the performance-monitoring log files from SPH and charted the servers' CPU and disk utilization for some customer. In Figure 2.8, the histograms of overall CPU and disk utilization for both types of systems are plotted. While the SQL servers primarily act as a backend for the ERP software and, thus, handle OLTP workloads, the BI servers were used to generate executive reports on a daily basis (OLAP). Figures 2.9 plot hourly CPU utilization over a period of one week. As the graphs indicate, the servers spend most of their time idle (below 20% CPU load) and are usually underutilized. Yet, at a few time intervals, peak performance is needed to fulfill all incoming request.

Overall, it gets obvious that the claims made by Barroso and Hölzle apply to these servers as well. Hence, typical servers are underutilized most of the time and waste huge amounts of energy due to their non-proportional power profiles.

Optimizing the energy consumption of server was therefore a recent hot topic in research and many different ideas were checked in order to reduce the power consumption of (database) workloads.

In [LP09], Lang and Patel present two techniques for a DBMS to trade performance for lower energy consumption: First, they took advantage of existing frequency and voltage scaling in modern processors, called *Dynamic Voltage and Frequency Scaling* (DVFS), and explicitly

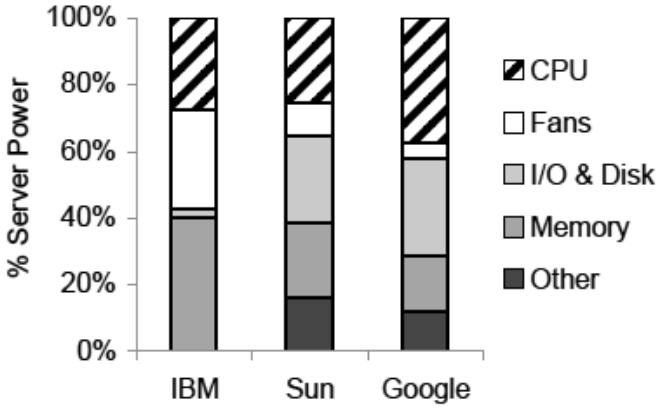


Figure 2.7: Server power breakdown, from [MGW09]

managed the speed of the CPU in order to save energy. With this technique, the authors were able to reduced energy consumption by ~5% while increasing runtimes by ~3%.

In the same publication, the authors delayed queries by collecting similar requests and bundling them into a single query plan, thus consolidating light workloads into a single, heavy-weight query. This technique, called *Query Energy-efficient by Introducing Explicit Delays* (*QED*), allowed energy savings of up ~50%, but also increased query response times by ~43%. By leveraging common access paths, the overall workload was reduced, which, in turn, reduced runtime and, thus, energy consumption. With these two techniques, the authors exemplify the possibility of trading energy efficiency for performance and vice versa. While these experiments reveal some potential, overall energy savings were marginal.

In [XTW10], the authors also explored power-performance tradeoffs in database systems. They tried to find query plans with lower power consumption by incorporating a power model into the optimizer. They found alternative plans with less energy consumption at the expense of higher query runtimes. By saving 10 – 20% of energy, query runtimes were prolonged over-proportionally. Hence, the authors conclude that no power reduction comes free of charge.

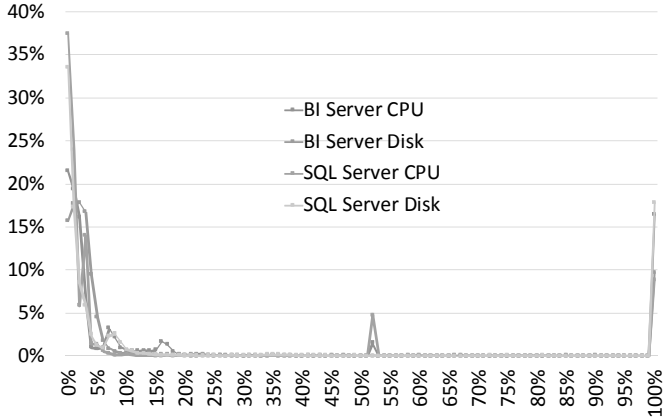
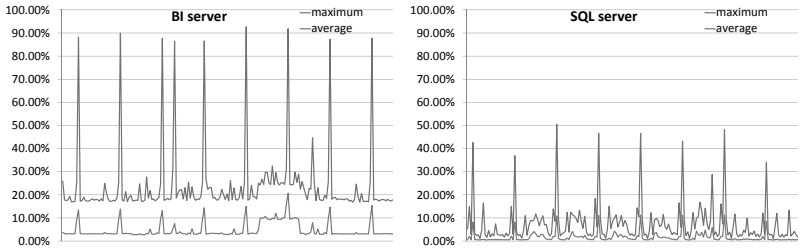


Figure 2.8: Average CPU and disk utilization of SPH's servers, from [SHK12]

In [Wan+11], Wang, Feng, Xue, and Song examined the possibilities to improve energy efficiency of a single server. They emphasize the need for more energy-efficient hardware and better power management controls. Also, they suggest creating power models to support "energy optimizers" that actively configure the system for energy efficiency. Their work is an exploration of probable options, divided into hardware- and software-based approaches. At the hardware side, they focus on the CPU as the main consumer of power—an assumption, that does not hold for database servers. They acknowledge the fact that current hardware has only limited potential to save power; thus, they emphasize the need for additional software-based approaches. As examples, they envision energy-aware optimizers and automatic resource consolidation.

In [LKP11], Lang, Kandhan, and Patel present a framework considering performance service-level agreements (SLAs) as well as energy-efficiency goals. In their work, they implement a DBMS optimizer that tries to meet performance goals while reducing the energy needed by explicitly controlling power and performance states of the underlying hardware, thus, overriding the kernel's power management. Hence, with their system, they are able to trade performance for energy savings,



(a) BI server (OLAP) (b) SQL server (OLTP)

Figure 2.9: Hourly CPU utilization of two servers, from [Sch11]

while satisfying given performance SLAs. In times of underutilization, the system was able to slow down and, thus, save energy.

To dynamically adjust a system to the current workload and, thus, reduce power consumption in idle times, Meisner, Gold, and Wenisch envision a system of blade servers, powered by a shared PSU [MGW09], called RAILS (*Redundant Array for Inexpensive Load Sharing*). They argue, in a shared-disk cluster of identical blades, the number of running blades can be fit to the workload and surplus nodes can be suspended to reduce overall energy consumption. While they simulated a workload of stateless, short running jobs and did not consider data management issues that naturally arise in such a cluster, this approach is a step in the right direction away from centralized systems exhibiting bad energy proportionality to a clustered solution with more fine-granular control.

In [PN08], Poess and Nambiar compare power consumption figures for recent TPC-C results. They identify power-intensive components and discover trends in power consumption over the last seven years of TPC-C runs. Further, they outline potential approaches to better energy efficiency. Since the TCP-C results were published without disclosure of power consumption data—a constraint first introduced by TPC-Energy [TPC10c]—the authors had to estimate power consumption of each of the components. They verified their model by comparing it with measurements on exemplary server setups.

The authors found out that peak power consumption of the system under test is steadily rising, from 2,000 watts in the year 2001 to 15,000 watts in late 2007. Likewise, energy efficiency—i. e., performance

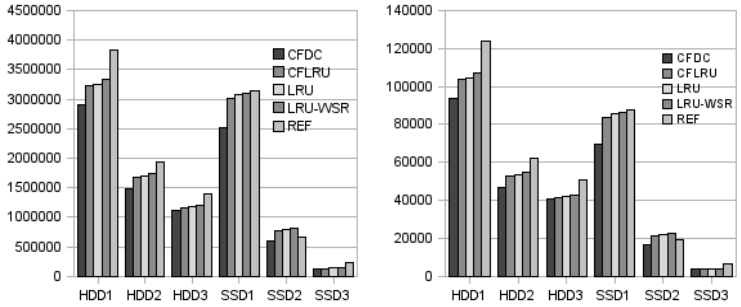
per watt—increased from 10 tpmC per watt to 30 tpmC per watt. Storage is the main power consumer in database systems, consuming about $\frac{2}{3}$ of the total power. The authors' investigation of power models and ways to integrate power-related measures into the TPC-* benchmarks ultimately resulted in the publication of TCP-Energy [TPC10c]. This benchmark is described in Section 4.1.4.

In [WTA13], Woods, Alonso, and Teubner demonstrated *Ibex*, an FPGA-accelerated query processing engine. They placed FPGAs between the storage layer and the database engine, enabling fast preprocessing on FPGAs to offload query operators from the main engine. While performance was their primary goal, they also reduced overall energy consumption per query.

Yi-Cheng Tu published various results from their work on energy efficiency in DBMSs [Tu+14; TWX11]. Their research prototype *Energy-Efficient Database Management System* (*E2DBMS*) annotates query plans with power cost to select the most energy-efficient alternative. Thus, the database optimizer is made energy-aware. To leverage the effects on query plan selection, additional ideas were proposed, e. g., consolidating workload on disk in order to reduce overall power consumption and to actively control power modes of hardware components (CPU, HDD) to trade performance for energy efficiency whenever possible. Yet, these suggestions are visions based on simulations, an actual implementation and evaluation is yet to come.

In our work[OHS10], we illustrated the close linkage of execution times and energy efficiency with his experiments with a single-server DBMS. All experiments were conducted in an identical system setting, i. e., mainboard, including processor and DRAM, IDE disk drive, memory size, OS, DBMS (except buffer management), and workload were left unchanged. For this reason, the details are not important in this context. The goal was to reveal the relationship concerning performance and energy use for different external storage media—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)—and buffer management algorithms[2].

[2]CFDC [OHJ09] optimizes page caching for SSDs. Here, we cross-compared CFDC to LRU, CFLRU [Par+06], LRU-WSR [Jun+08], and REF [SS08], some of which



(a) Execution times (ms)

(b) Energy use (J)

Figure 2.10: Performance and energy consumption running the TPC-C trace, from [OHS10]

Our primary goal was to study DB buffer management and to reveal the role of the external storage used and the influence of algorithmic optimizations. We implemented the DBMS storage system with a number of known algorithms for the DB buffer. Our main concern was to examine how well these algorithms perform on conventional magnetic disks and on SSDs. Yet, we also evaluated, how much energy is used for buffer management in either case, i. e., how energy efficient are these algorithms in differing environments?

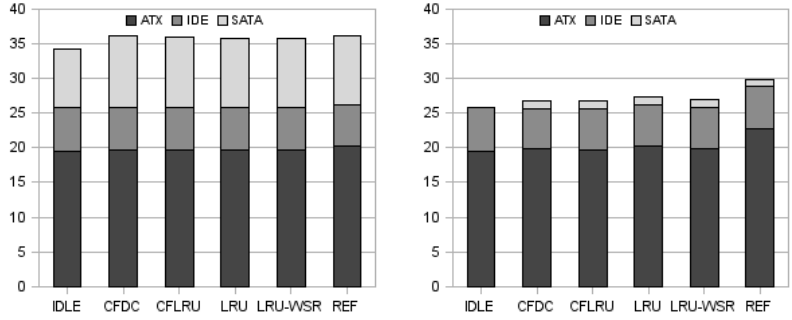
Here, we want to repeat these answers given in [OHS10], thereby preparing our arguments for energy-proportional DBMS management. To represent a realistic application for our empirical measurements, we recorded an OLTP trace (a buffer reference string using a relational DBMS) of a 20-minutes TPC-C workload with a scaling factor of 50 warehouses. As test environment, we used an Intel Core2 Duo processor and 2 GB of main memory (denoted as ATX). Both the OS (Ubuntu Linux, version 2.6.31) and the DB engine were installed on an IDE magnetic disk (system disk). The test data (as a DB file) resided on a separate magnetic disk/SSD (data disk, denoted as SATA). The data disks represent *low-end* (HDD1/SSD1), *middle-class* (HDD2/SSD2), and *high-end* (HDD3/SSD3) devices. They were connected to the sys-

are also tailor-made for SSD use.

tem one at a time. We ran the TPC-C trace for each of the five algorithms under *identical* conditions and repeated it for each of the devices under test. Using a buffer size of 8000 pages (64 MB), we minimized the influence of the device caches. However, not the absolute but the relative differences are most expressive. Execution times and related energy use in Figure 2.10 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.[3] If we compare the results among devices of the same class, CFDC turns out to be the best performing algorithm, i. e., it also provides superior performance when used for HDDs or sets of heterogeneous devices.

The key effect identified by Figure 2.10 is further explained by Figure 2.11, where the breakdown of the average working power of hardware components of interest 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 [THS10] 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*". Furthermore, no clear difference can be observed between the various algorithms, although they have different complexities and, in fact, also generate different I/O patterns. This is due to the fact that, independent of the workload, the processor and the other units of the mainboard consume most of the power (the ATX part in the figure) and these components are not *energy proportional*, i. e., their power use

[3] The difference between the execution times of the algorithms becomes smaller on SSD3, (see Figure 2.10), because its I/O is much faster than on other devices, yielding the buffer layer optimization less significant, and SSD3 has supposedly the largest device cache, since it is the newest product among the devices tested.



(a) HDD3 (b) SSD3

Figure 2.11: Breakdown of average power (watts)

is not proportional to the system utilization caused by the workload.[4] Main-memory power usage is more or less independent of system utilization and increases linearly with the memory size, i. e., the number of RAM modules. In our case, we only had a single 2GB module. Increasing the main-memory size by adding more RAM modules would rapidly shift in our scenario the relative power use close to 100%, even in the idle case. The breakdown of the average working power in Figure 2.11 reflects the average system utilization obtained for individual trace executions. If we evaluate how energy consumption depends on system utilization, we roughly get for our configuration—with a single computing node—the characteristics sketched in Figure 2.12.[5]

As indicated, the scope for optimizing the relative energy efficiency by software means is limited and would almost disappear when a large memory is present. But this scope could be widened, if hardware optimizations could be invented (e. g., reduction of RAM's energy consumption). Using a *single computing node*, we would never come close to the ideal characteristics of energy proportionality. Note, we cannot just switch off RAM chips, especially in the course of DBMS process-

[4] The elapsed time T of the workload almost completely determines its energy consumption E (note, $E = \bar{P} \cdot T$, where $\bar{P}$ is the average power measured).

[5] Memory of enterprise server quality consumes ∼10 W per 4GB DIMMS [THS10].

ing, because they have to keep large portions of DB data close to the processor. Preserving this reference locality is the key objective of each DBMS buffer.

With our current flash-based optimizations, we do not obtain any noticeable effect on overall energy saving – except for continuous peak load situations. Given normal load patterns and arrival times, an average request is processed more efficiently, i. e., system resources are allocated for shorter intervals, thereby reducing system utilization even further.

In summary, energy saving is impressive, if we consider the experiment in isolation where system utilization is steadily kept very high, i. e., $> 90\%$.

In his PhD thesis [Ou12], Yi Ou examined the performance and energy saving potential of SSD-optimized buffer algorithms. By replacing traditional storage disks (HDDs) with SSDs, performance and energy efficiency can already be increased. Optimized, SSD-aware algorithms leverage internal properties of SSDs and deliver even more performance, which results in turn in better energy efficiency. Yet, Yi Ou concludes that his algorithms would exhibit even more power savings when run on more energy-proportional devices.

In summary, all studies based on single servers either trade performance for power savings or exhibit that the best performing configuration is also the most energy-efficient one, as discovered by Tsirogiannis, Harizopoulos, and Shah [THS10].

In Figure 2.12, 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. Of course, the server needs the most power at high utilization, but it already consumes $\sim$50% of its peak power when idle. With increasing workloads, power consumption quickly converges to its peak. Even low workloads require a disproportional high amount of power.

As outlined, about half the power of today's hardware is consumed statically and there is no way to eliminate or reduce this using software solutions. By optimizing the hardware configuration to fit a certain task, power consumption can be reduced, by the price of becoming inflexible to changing workloads. As revealed earlier, typical utilization patterns cover a broad range of workloads and (DB) servers need to be laid out to handle even peak utilization.

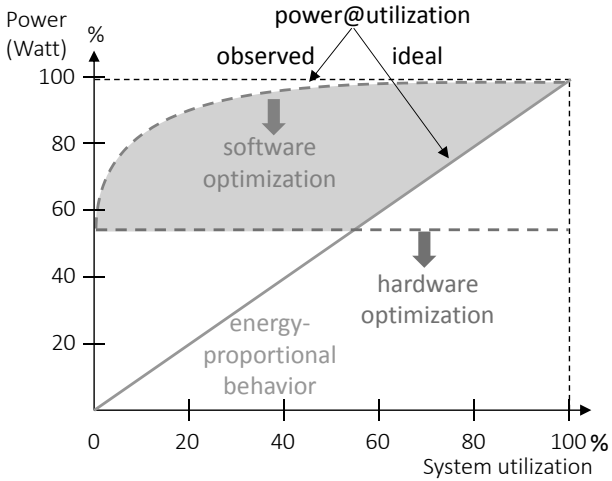


Figure 2.12: Power use over single-node system utilization

For all other workloads, this requirement will result in bad energy efficiency. Therefore, all single-server-based solutions can only be energy efficient when fully utilized.

2.1.3 Amdahl-balanced components

Amdahl observed that the overall performance of a computer system is determined by its slowest component [Amd13; Sza+10]. Therefore, it is necessary to carefully select all components to match each others throughput. The best configuration is dependent on the workload, but, in general, throughput of the I/O subsystem must match the system's processing power. Further, for distributed systems, the interconnecting fabric must handle the desired throughput produced by the nodes in the cluster. Replacing a single component with a faster one can only yield throughput improvements, if the other components do not form a bottleneck. For example, replacing the CPU with a faster one will only increase overall performance if the original CPU was the slowest component in the system. Other factors, as network bandwidth and disk latency, may quickly impose constrictions and slow down data transport

to the processing unit. Hence, overall speedup is best if all components are carefully selected. Especially in terms of energy efficiency, it is important to balance a system. Faster components usually come with higher (static) power consumption, which needs to amortize by faster query runtimes.

2.2 Distributed Systems

Studies on centralized systems did not reveal any huge energy saving potentials, mostly due to the fact, that today's hardware is still not energy proportional and the variable operating modes of components do not impact overall energy consumption much. Therefore, focus shifted from single servers to distributed systems, where each system can be seen as a component of the cluster and turned on and off independently.

In [And+09], the authors introduced a key-value store, running on a cluster of lightweight nodes, called *FAWN (Fast Array of Wimpy Nodes)*. By choosing SSDs as storage and low-power components, each node consumes little energy. FAWN implements a ring architecture, using chain replication [RS04] to increase fault tolerance. Their implementation support transaction-less put/get operations on small values and performance scales very well with the number of nodes in the system. Yet, dynamic aspects are not implemented.

In [Pin+01], the authors examined a cluster of WWW servers, answering stateless, read-only requests. Each request was handled by a single server and no coordination was necessary, due to the simplicity of the workload. To adjust the size of the cluster to a given load, the authors implemented a way to automatically scale out by adding additional servers and to scale back in respectively, by removing servers from the cluster and shutting them down. This is an interesting approach, applied to a non-database related field, which delivered promising results and quick reaction times to workload shifts. Yet, the main concern in databases—data migration—is not addressed with this solution and remains an open issue.

Lang, Patel, and Naughton introduced a similar, data-centric approach in [LPN10a]. They replicated data in a cluster along a chain of nodes using a technique called *Chained Declustering* [HD90]. With this technique, one node holds the primary copy of a partition, while

the node next in the chain (arbitrary ordering) holds a hot copy. Hence, both copies can serve read-only requests. To enable energy management in a cluster of nodes, they selectively shut down superfluous nodes at low utilization, reducing overall energy consumption while concentrating query evaluation on the remaining nodes. Yet, their technique is limited to scale down to 50% of the initial nodes at max, otherwise, partitions start to get inaccessible. Other open questions are the rebalancing of partitions in case of skewed workloads and the support of updates, especially when an update of an offline replica is considered. Parts of the latter question are answered in [LPN10b].

After evaluating read-only workloads, Lang, Patel, and Shankar examined wimpy, e. g., lightweight nodes in [LPS10] and analyzed a cluster similar to the *FAWN* cluster introduced in [And+09]. In the original publication, the cluster was benchmarked with lightweight workloads. The authors posed the question, whether a cluster, consisting of wimpy nodes, is able to process complex workloads, too. Therefore, they simulated runs of heavy-weight queries on different kinds of nodes and compared performance and energy efficiency. They conclude that, due to friction losses and coordination overhead, performance does not scale linearly with the number of nodes and challenging workloads better run on smaller clusters of brawny servers, instead of a large cluster of wimpy ones.

Based on these conclusions, the authors ran more experiments on clusters of mixed-size nodes to determine the best configuration for various workloads [Lan+12]. They could show that careful selection of the hardware makes a big difference in performance and energy consumption, with respect to the query. In their work, they also examined the scalability of parallel DBMSs and concluded that easily partitionable workloads exhibit almost linear performance improvement on scale-out architectures. On the other hand, partition-incompatible access patterns exhibit very bad performance in a distributed environment. Non-scalable workloads therefore prefer beefy servers, instead of multiple wimpy nodes. Yet, the most interesting finding in this work is, that *the best-performing configuration is not always the most energy-efficient one*—a conclusion contradictory to the findings on centralized machines in [THS10].

While the work of Lang, Patel, and Shankar compares statically configured clusters and does not tackle the issue of automatically adapting

to the current workload, their findings are valuable for our research. They show that the optimal size of the cluster depends on the workload and that energy efficiency and performance can be traded against each other.

2.2.1 Distributed and energy-efficient storage

With the emergence of distributed computing, underlying storage architectures became distributed as well. In shared-nothing databases, disks are attached directly to the nodes, which can locally process queries assigned to them. Hence, storage distribution is predefined, but the right partitioning is an open question.

On the opposite, shared-disk DBMSs have different distribution networks for processing and storage. Yet, centralized storage solutions cannot serve the plethora of I/O requests coming from a multitude of servers. Hence, storage scale-out is a necessity for modern, distributed database systems.

Placement of data in a shared environment is a performance-critical issue. [MD97] analyzes data placement issues in a shared-nothing DBMS cluster. The authors develop a new placement strategy by employing simulation of access patterns. They analyzed the impact of workload shifts and determined the right degree of declustering to maximize performance. Further, the authors analyzed the placement of partitions among nodes and the effects on the system's behavior.

In [Ver+10], the authors implement a storage virtualization layer on a single server to transparently migrate data blocks among disks, called *SRCmap*, which stands for *Sample, Replicate, Consolidate Mapping*. By dynamically placing blocks on an optimal configuration of disks, the number of disks required to satisfy a given workload could be minimized, reducing overall power consumption. To reduce the impact of updates, *write offloading* is used, redirecting write operations to spare disks. Their experimental evaluation reveals near-energy-proportional behavior under static workloads.

You, Hwang, and Jain have published their work on data partitioning in [YHJ11]. Similar to *SRCmap*, they implemented a framework that divides the dataset into hot and cold data and tries to optimize the placement on disks, called *Ursa*. Their focus did not lie on energy efficiency, but rather on dynamic migration of data to react to shifting

workloads. Their framework was managing a large scale (1000s) of nodes, dynamically moving ownership of objects among them. Their aim was to minimize the number of nodes while keeping performance goals. Yet, their approach migrates self-contained logical objects, with no interactions among them.

2.2.2 Dynamic clustering

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.

In his PhD thesis [Das11], Sudipto Das implemented an elastic data storage, called *Elastras*, able to dynamically grow and shrink on a cloud. As common in generic clouds, his work is based on decoupled storage where all I/O involves network communication. *Key Groups*, application-defined sets of records frequently accessed together, can be seen as dynamic partitions that are often formed and dissolved. By distributing the partitions among nodes in the cluster, both performance and cost can be controlled.

A lot more data management systems working on a cloud have been proposed. In [Bra+08], Brantner, Florescu, Graf, Kossmann, and Kraska designed a DBMS using *Amazon S3* as storage and running on top. Lomet, Fekete, Weikum, and Zwilling divided the database into two layers, one transactional and one persistence component that can run independently [Lom+09].

In [Arm+09], Armbrust, Fox, Patterson, Lanham, Trushkowsky, Trutna, and Oh proposed a scalable storage layer supporting consistency and dynamic scale-out/in called SCADS. Objects in SCADS are stored in logical order. Hot, i. e., frequently accessed objects are distributed among disks to improve access latencies and mitigate bottlenecks. The system was also extended to automatically adjust to workload changes and autonomously redistribute data.

Besides relational approaches, other implementations relax traditional DBMS properties to gain performance and simplify partitioning.

Yahoo PNUTS [Coo+08], Bigtable [Cha+06], and Cassandra[6], are examples of systems sacrificing transaction or schema support and query power [Arm+09]. Instead of arbitrary access patterns on the data, only primary key accesses to a single record are supported [VCO10].

As an improvement, Amazon's SimpleDB[7] allows transactions to access multiple records, but limits accesses to single tables. Moreover, most current scalable data storage systems lack the rich data model of an RDBMS, which burdens application developers with data management tasks. Yet, no *fully autonomous, clustered DBMS* exists which can provide ACID properties for transactions and SQL-like queries while dynamically adjusting its size to the current workload.

2.2.3 Database partitioning

Partitioning a table is an old concept and widely used. Splitting tables into multiple partitions has mainly two advantages:

First, by dividing the table into smaller logical groups—either by key ranges, hash, or based on time intervals—the amount of data necessary to access for a particular query can be reduced. All major DBMSs use techniques like *partition pruning* or *partition-wise joins* to reduce the amount of data to be read for a query [Ora07]. DB partitioning also enables parallelization and, thus, better utilization of the hardware and higher performance by parallelizing data accesses. This is especially true for a distributed DBMS, where partitions can be allocated to various nodes, enabling processing on more CPU cores and—in contrast to single-node databases—also bringing in more MMUs[8], main memory, and storage disks to support query processing.

Second, partitions can be used as units of logical control over the data contained, i. e., the node owning a partition is responsible for its integrity and concurrency control. By dividing a large table into smaller partitions, the resulting control overhead can be shared among nodes. Hence, instead of a single node having to manage an entire table with

[6] `http://cassandra.apache.org/`
[7] `http://aws.amazon.com/simpledb/`
[8] MMU = Memory Management Unit, providing additional bandwidth to resolve the bottleneck between CPU and main memory.

long request queues waiting for locks, to perform a variety of integrity checks, and to serialize processing due to log writes, all tasks can be split up into partitions and maintained by a group of nodes.

Partitions provide a logical encapsulation over a group of records, i. e., records do not span multiple (horizontal) partitions. Hence, moving a partition from one node to another does not affect other parts of the table. For this reason, DB partitioning is an ideal candidate to provide the building block for dynamic reorganization in our DBMS.

In his dissertation [Das11], Sudipto Das implemented a distributed DBMS as introduced earlier. To partition the data among nodes, he introduced *Key Groups*, a temporary set of data items that are jointly accessed, also forming transactional boundaries. Key groups are formed dynamically upon client request and predefine access patterns and consistency rules. Yet, grouping does not affect physical storage of items; underneath, a shared-disk system is providing all data.

Pandis, Tözün, Johnson, and Ailamaki proposed a *physiological partitioning* scheme in the context of a multi-threaded DBMS ([Pan+11]). They identified two existing techniques, physical and logical partitioning. Logical partitioning corresponds to a shared-everything approach (as classified by the authors [Töz+13][9]), whereas physical partitioning is equivalent to the data distribution of a shared-nothing DBMS. In their work, they introduced multi-rooted B*-trees, each identifying a partition of a DB table. By allowing only a single thread at a time to access such a tree, they eliminated contention and locking overhead.

In [Töz+13], the authors introduced a dynamical load balancing mechanism (DBL) that detects and resolves load imbalances. Imbalances require changes in the partitioning scheme, which are automatically performed in the system and partitions can be split up and merged to spread out and consolidate load. Their algorithm tries to minimize change operations to reduce interference with query evaluation. While they focused on a single node with multiple CPU cores to assign partitions to and do not focus on energy efficiency, their concept of dynamically reassigning partitions in order to load-balance the system looks promising and can be transferred to shared-nothing database nodes.

[9] Using the traditional classification [Rah93], it is rather a shared-disk approach.

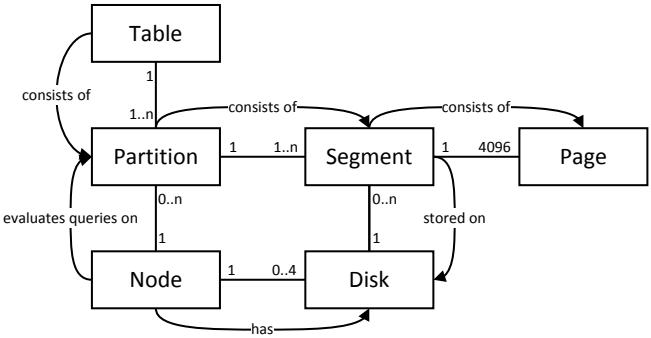


Figure 2.13: Database schema

2.2.4 Partitioning examined

Related publications concerning dynamic partitioning aspects have already been outlined earlier in 2.2.3. To clarify the concepts and to put them into perspective with this work, we are distinguishing between three implementations: Physical, logical and physiological partitioning.

Each of the three approaches exhibits certain advantages and disadvantages in data partitioning, which we will outline shortly. Especially in the context of an elastic DBMS, where the partitioning scheme is subject to change over time, each of them offers different potential.

Definitions

Before discussing different partitioning schemes, we need to clarify the terms used, starting from storage structures and moving upward the database layers. Figure 2.13 clarifies these terms and their relationships.

Node Every machine in the cluster is called a *node*, typically attached with one or more storages and able to evaluate queries.

Storage A *storage* is providing persistent, online storage space to the DBMS. Hence, storage may either refer to HDDs or SSDs. Other forms of persistent storage, e. g., tapes or PCM are not considered here. Nodes maintain exclusive access to storages directly connected, but may allow

remote nodes to access their data. Yet, the nodes directly connected are responsible for coordinating read and write accesses to their storages.

Block A *block* is the smallest addressable unit of data on a storage. In our implementation, the block size is 8 KB (8192 Bytes). All I/O operations are carried out at this granule, even if only parts of the data are required to be read/written. At the storage level, each block can be identified by its address, consisting of three parts: *NodeID, StorageID and BlockID* (*NSB*). This identifier is unique in the entire cluster and specifies exactly one storage block.

Segment A *segment* consists of 4096 blocks, stored consecutively on disk, thus having the size of 32 MB. Segments are used for bookkeeping and maintenance to abstract from a fine-grained block-based mapping. Segments are the building block of bigger, logical database entities in WattDB, i. e., partitions. They describe large-grained units of storage that can be used to map from logical to physical addresses. A more flexible mapping from logical pages to physical blocks in a 1:1 manner would require much more main memory to represent such a mapping.

Page The data granularity inside the database buffer is a *page*. It has the same size as a storage block, 8 KB. To differentiate between logical pages and physical data blocks, the two terms are used respectively.

Pages are also the unit of data transfer between nodes. By defining all pages and blocks of equal size, buffer management and page propagation are very simple to implement.

Partition A *partition* is the mapping between logical records and their physical representation in WattDB.

It consists of at least one *segment*, denoting the storage location of all records within the partition. A partition holds the mapping of logical pages to physical segments (and blocks within). Each segment is located on a specific disk on a node in the cluster. Segments stored on the same node as the partition do not require network access to fetch data, but accessing only local disks may impose an I/O bottleneck for the partition. Therefore, it is also possible to remotely address segments, stored on other nodes. Assignment policies of segments to partitions

depend on the partitioning scheme used, i. e., whether to allow access to segments stored on remote nodes (shared disk) or not (shared nothing). Based on their assignment to nodes and disks, access costs to segments vary.

Partitions hold logical records of data. By default, they are index-organized [Sri+00] w. r. t. the primary key with support for additional, secondary indexes.

Nodes have exclusive ownership of partitions, making them responsible for query evaluation, integrity control, and access synchronization.

Index In WattDB, *indexes* are realized using B*-trees [BM70; Bay71] 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. For each partition, a primary-key index is automatically created, allowing index-organized data access. Secondary indexes can be created manually to support query evaluation.

Primary indexes are always stored in the same partition as the data they are indexing. For specialized partitioning schemes, introduced later in this thesis, placement of indexes may be narrowed down further.

Table A DB *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*[10], each belonging to a specific node, responsible for query evaluation, data integrity (logging), and access synchronization (locking). The partitioning scheme used to divide tables into partitions is application-dependent, as some applications may benefit from distinct key ranges, while others may prefer scattered data. In the following, we introduce three partitioning schemes we are going to use in this thesis, based on the concepts on physiological partitioning from Tözün, Pandis, Johnson, and Ailamaki in [Töz+13]. While the basic concepts are explained in this Section, an evaluation of all three schemes and their use for elastic clustering is examined in Chapter 9.

[10] `http://dev.mysql.com/tech-resources/articles/partitioning.html` outlines horizontal partitioning.

Physical partitioning

Physical partitioning is the most straightforward partitioning scheme. It operates at the data access layer and does not involve logical access paths. The optimization layer is therefore oblivious of the segment distribution at the storage layer. To implement physical partitioning, the database (storage) is divided into partitions, and each partition is stored on specific disk(s). Partitions are not segmented into distinct key ranges, records may be stored in any partition.

As the name suggests, physical partitioning does not segment data based on logical properties, but rather divides the storage space into segments, stored in distinct locations, to parallelize access to database pages.

Physical partitioning already offers load balancing and parallel query evaluation, but without knowledge of the data stored inside, further optimizations by higher DB layers, e. g., partition pruning and partition-wise joins, are thwarted.

Logical partitioning

Logical partitioning divides the dataspace into units by logical predicates. Hence, instead of blindly chopping tables into parts as with physical partitioning, *logical partitioning* splits data by its intrinsic properties, i. e., the record's attribute values. In WattDB, data is partitioned by primary-key value ranges.

Since partitioning is performed by logical attributes, higher database layers may exploit the partitioning scheme to optimize query evaluation.

Figure 2.15 sketches a logically partitioned table with two partitions, consisting of segments, possibly allocated on remote nodes. Each partition holds records from distinct primary-key ranges. Hence, the query optimizer may utilize this information to direct point queries to the respective partition.

Physiological partitioning

Physiological partitioning was introduced in [Töz+13] to assign data partitions exclusively to CPU threads, thus eliminating locking and reducing contention. Their approach is explained earlier. With this ap-

proach, logical allocation (ownership) and physical storage of records are separated, hence *physio-logically* partitioned.

Similar to the original approach, physiological partitioning among nodes encapsulates key ranges in partitions and assigns them exclusively. While physiological partitioning originally assigns partitions to CPU cores to eliminate contention, we assign partitions to nodes for the same reason.

Partitions consist of segments, like in the previous approaches.

In contrast to logical partitioning, each single segment keeps a primary-key index for all records within it. Hence, partitions only contain a meta-index on top, keeping information about key ranges in the attached segments. This top index is very small compared to an index containing all records from all segments.

Figure 2.16 sketches the design of physiological partitioning. Two partitions on different nodes are shown, both consisting of several sub-partitions, contained in segments. Primary-key ranges for each of the mini-partitions are outlined.

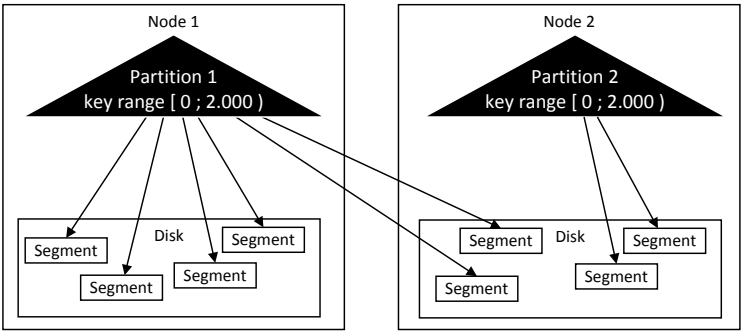


Figure 2.14: Physical partitioning

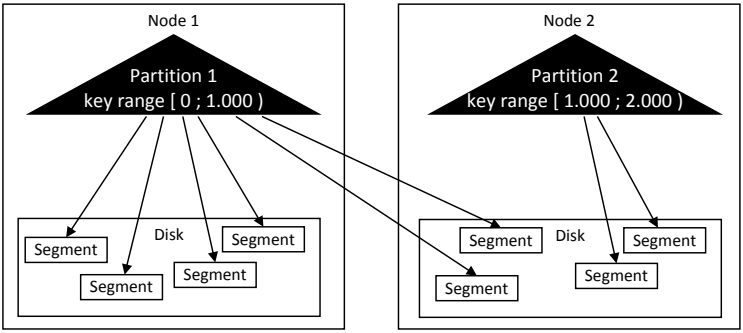


Figure 2.15: Logical partitioning

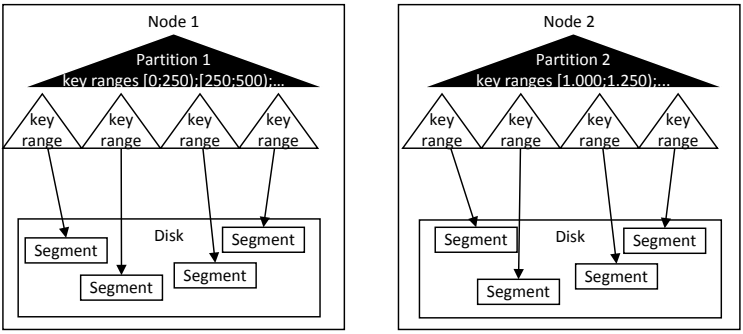


Figure 2.16: Physiological partitioning

3 Measuring Energy Consumption

All computer devices require electricity to function. To quantify their energy consumption, the measurement units and the basics of measuring energy and power consumption have to be clarified first. Since the terms *power* and *energy* have similar, but ambiguous meaning, it is important to define both.

Exactly measuring energy consumption of a database cluster over long time periods is non-trivial. Existing energy measurement devices, are first explained and evaluated for their applicability to our use-case. Afterwards, a custom-made monitoring framework is presented, since existing measurement solutions did non quite cover our needs.

3.1 Metrological Background

Here, basics about measuring electrical power and related units are given.

3.1.1 Charge

Electrical charge is a fundamental physical unit. Its formula sign is Q (lat. "quantity"). Charge is expressed in units of *coulombs* (C), where 1 coulomb is equal to the charge of approximately $6.241 * 10^{18}$ electrons (or protons for positive charges, respectively) [BIP06]. All electrical effects stem from the movement of charges.

3.1.2 Voltage

Voltage, or *electrical potential difference*, is measured in units of *volts* (V). It quantifies the difference in electrical potential of two points.

Since the English notation is ambiguous and uses the symbol V for both, the unit and the formula symbol of voltage, this work adheres to the German convention and denotes U for the formula symbol and V for the unit of voltage [DIN94]. Based on SI-units, voltage is defined as follows [BIP06]:

$$1V = 1\frac{Joule}{Coulomb} = \frac{kg * m^2}{s^2 * C}$$

Voltage is always measured between two points, therefore, a reference point called *ground* is used to act as a null-potential for all measurements. By definition, voltage of any other point is then expressed as potential difference against ground.

Voltage alone does not describe electricity appropriately, but it is a prerequisite for electrical charges to move and do work.

3.1.3 Current

Electrical current describes the flow of electrical charges between two points. An electrical current will only flow between points with potential difference (voltage difference). The formula symbol for current is I, the unit *ampere* A [BIP06]. 1 ampere equals 1 coulomb per second, hence:

$$1A = \frac{1C}{1s}$$

3.1.4 Alternating current

The previous definition of current assumes an unidirectional flow of electrical charges, called direct current; Negative charges (electrons) move from anode (denoted with a minus sign: -) to cathode (denoted with a plus sign: +).

In *alternating currents*, charges are periodically reversing the direction of flow. Special care must be taken when calculating derived values, e. g., power and work from alternating current, since the momentary current is changing rapidly.

3.1.5 Resistance

Connecting two points with potential difference will lead to a current flow. In theory, without any electrical resistance between the points,

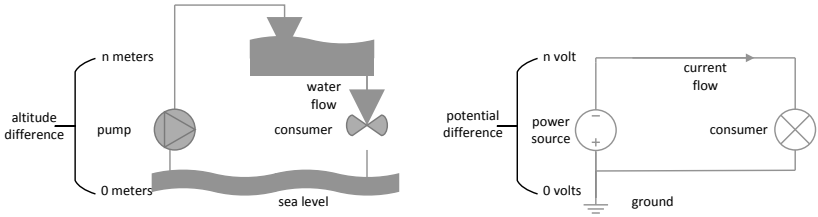


Figure 3.1: Analogy between water and electric circuit

the current flow will go towards infinity and the potential difference will shortly be cleared. In practice, all connecting wires and every every electrical device provide resistance and reduce the current flow.

Electrical resistance is denoted by the symbol R and defined as the quotient between current and voltage:

$$R = \frac{U}{I}$$

Resistance is measured in units of *ohms* denoted by the symbol Ω:

$$1\Omega = 1\frac{V}{A}$$

3.1.6 Water circuit analogy

The connection between the measures introduced earlier can be clarified easily with an analogy between an electrical circuit and a water circuit as sketched in Figure 3.1 [Nav14]. Both, electrical and water circuits have a reference point towards all water/charge flows in the diagram. For the water circuit, it is sea level, for the electrical circuit, it is ground. The definition is arbitrary, water/current can also flow down from the reference point to another (deeper/lower voltage) point.

In the water circuit, a pump presses water upwards to a higher level, giving it *potential energy*. The higher the water is pumped, the more potential energy it contains. A power source, e. g., a battery, can similarly induce potential differences to electrical charges. As a result, there is a potential difference before and after the pump/battery.

That difference in (electrical) potential is the reason for the following flow of water/charges down to the ground, which can perform work while flowing. An intermediate consumer, transforming the energy of the water/electricity into other forms, will slow down the flow due to its resistance.

3.1.7 Power

Power (consumption) is the rate, at which energy is transferred, its formula symbol is P. Electrical power is linearly dependent on the potential difference and the electrical current flow [HRW10]:

$$P = U * I$$

Power is measured in units of *watts* (W). In electrical engineering, 1 watt expresses a current flow of 1 ampere through an electrical potential of 1 volt:

$$1W = 1V * A$$

Since current can be expressed by voltage and resistance, the formula can be rearranged to:

$$1W = 1\frac{V^2}{\Omega}$$

When measuring power consumption under alternating currents, the momentary voltage and current flow need to be multiplied. Simply averaging voltage and current over a measurement period will lead to invalid results, as illustrated in Figure 3.2, depicting the course of some alternating current, voltage and the results (absolute) power consumption. The averages of current and voltage over time equal zero, hence, the resulting power would be incorrectly calculated as $0V * 0A = 0W$. In the appropriate calculation, momentary values of current and voltage need to be multiplied, resulting in a power consumption $\neq 0$.

3.1.8 Work

Work, or energy consumption, is defined as power consumption over time:

$$W = P * t$$

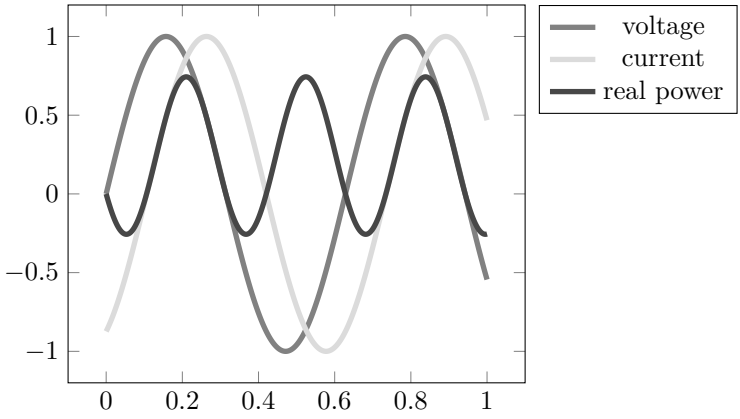


Figure 3.2: Voltage, current, and real power in AC

The formula symbol of (electrical) work is W, its unit is joules (J), defined as follows [ABS06]:

$$1J = 1W * 1s$$

The formula above is valid for constant power consumption during the time interval t. For variable consumption, an integral over time is needed:

$$W = \int P_t dt$$

The energy consumption of any device can thus be calculated by measuring current and voltage during a given period.

3.2 Consumption of Electricity

In recent years, electricity consumption has strongly increased in the US and worldwide: As of 2007, US datacenters consumed about 1.5% of the country's total electricity generation, projected to 3% in the next years [EPA07]. For 2010, datacenters in the US used between 1.7% and 2.2%—varying from study to study. Worldwide, datacenters currently consume about 1.1% to 1.5% of all electricity generated [Koo11]. Figure

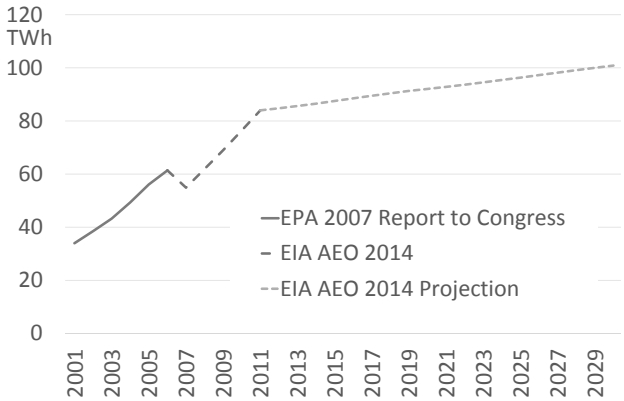


Figure 3.3: US datacenter electricity use in TWh ($= 10^9$ Wh). Data from 2001 to 2006 are estimates from the EPA Report to Congress 2007 [EPA07], data from 2007 to 2011 are from EIA's Annual Energy Outlook 2014 [EIA13], and data from 2012 to 2030 are projections based on the annual growth in electricity use in the commercial sector as estimated by EIA's Annual Energy Outlook 2014 [EIA13]

3.3 plots historic and projected energy consumption of US datacenters from 2001 to 2030. From 2000 to 2005, worldwide datacenter electricity consumption doubled. Contrary to older projections, from 2005 to 2010, the rate of growth slowed down due to the economic crisis (starting 2008). Additionally, the emergence of cloud computing and efforts to improve energy efficiency since 2005 further diminished the growth. In contrast to earlier projections, the main reason for this stagnation were not improvements in energy efficiency, but rather less new server installations than predicted [Koo11].

Although the consumption figures (historical and projected) are educated guesses, all estimates predict electricity consumption to rise continuously. With the rapid increase in datacenter numbers and sizes, electricity consumption will go up, even with better energy-efficient hardware [Koo11].

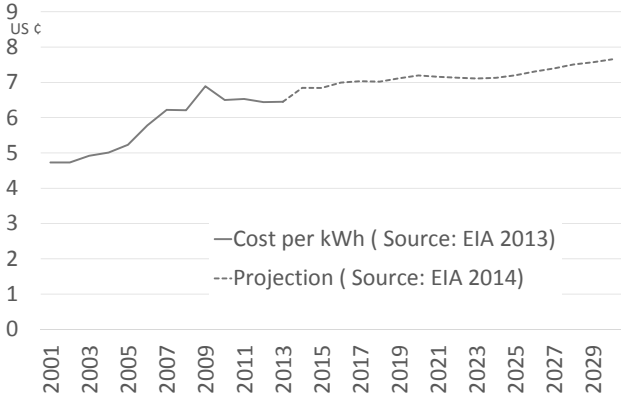


Figure 3.4: US industry-sector electricity price per kWh. Historical prices [EIA13] and projected development [EIA14]

3.3 Cost of Electricity

While the demand for electricity has steadily risen, the cost of electrical energy has increased as well: According to EIA, electricity price per kilowatt-hour for industrial use has risen from 4.73 US¢ in 2001 to 6.53 US¢ by the end of 2011 and 6.62 US¢ by the end of 2013. Hence, in 10 years, the price increased by almost 40%. Although price increases after 2011 were not as steep any more, again, due to the economic crisis, future projections expect the cost of electricity to go steadily up [Koo11]. Figure 3.4 plots historical prices as listed in EIA's Annual Energy Outlook 2013 [EIA13] and projected price trends in the next years, as predicted by EIA's Annual Energy Outlook 2014. As the graphs suggest, electric power is expected to become more and more expensive within the next years.

Similar trends are noticeable in the european union, where prices for electrical power have constantly risen over the last years—for example from 0.101 €/kWh in 2008 to 0.126 €/kWh in 2013, an increase of almost 25% in five years [Eur14].

The rising consumption and prices of energy led to various approaches to reducing overall energy consumption and to better utilize electricity. For example, new datacenters are placed in environmentally advanta-

geous regions, i. e., Microsoft opened a new center in Finland, Facebook built a new datacenter in Sweden, and the NSA invested in a complex in Utah's mountain-side [Kon13]. To put the waste heat, generated by cooling the servers, to good use, lots of datacenter operators have put them to good use to heat nearby buildings or facilities.

Yet, reducing overall energy consumption in the first place became a major goal in all areas of IT.

3.4 Energy Efficiency

The Oxford Dictionary defines *efficient* in a technical sense as "[..] achieving maximum productivity with minimum wasted effort or expense" [Ste10]. Hence, for computer systems, efficiency describes the ratio of processed data to total resources needed for processing. There exist several efficiency metrics, which systems can be tailored to. The two most common are **time efficiency**, which requires the system to take a minimal amount of time for processing, and **memory efficiency**, which rates efficiency based on the algorithms' memory needs [BM99]. Therefore, *energy efficiency* describes the ratio of a system's useful work to its total energy consumption taken in. A system's energy efficiency is higher, if either the same task is achieved with less energy consumption (fixed work budget) or by achieving more work with the same amount of energy (fixed energy budget). In databases, efficiency is rated in transactions per joule:

$$energy\ efficiency \quad = \quad \frac{\#\ of\ transactions}{energy\ consumption}$$

which can be transformed to the amount of work done per time unit when a certain amount of power is given:

$$energy\ efficiency \quad = \quad \frac{tps}{watt}$$

The higher the energy efficiency, the better a given system transforms electricity into "work". Note, this is the inverse of the formula used in TPC-Energy which applies watts per tps as its metrics. The rationale of the TPC for choosing the inverse was the desire to be similar to the traditional TPC metrics *price per throughput* and, furthermore, to allow a secondary metrics for each of the subsystems [SHK12].

3.5 Energy Proportionality

Energy efficiency is a measure to rate either the efficiency of a single server performing various tasks at different configurations or to benchmark competing systems with a given energy budget. Efficiency is always relative, since there is no absolute way of telling, whether a task is done efficiently without comparison to other systems, performing the same task.

In addition to that absolute measure, Barroso and Hölzle introduced the term *energy proportionality* [BH07]. Energy proportionality is defined as the ratio of a system's energy efficiency to it's peak efficiency (at 100% utilization). Ideally, the power consumption of a system should be determined by its utilization [Här+11]. Energy proportionality describes the ability of a system to scale its power consumption linearly with its utilization.

Therefore, energy proportionality can not be expressed using a scalar value. Instead, a function or graph is needed to display the characteristics of a system. For each level x, $0 \leq x \leq 1$, of system utilization[1], we can measure the power used and denote this value as the actual power consumption at load level x . To facilitate comparison, we use relative figures and normalize the actual power consumption at peak load ($x = 1$) to 1, i.e., $PC_{act}(x = 1) = 1$. Using this notation, we can characterize a system whose power consumption is constant and independent of the actual load by $PC_{act}(x) = 1$.

Note, we obtain by definition true energy proportionality at peak load, i.e., $PC_{ideal}(x = 1) = 1$. In turn, a truly energy-proportional system would consume no energy when it is idle (zero energy needs), i.e., $PC_{ideal}(x = 0) = 0$. Due to the linear relationship of energy proportionality to the level of system utilization, we can express the ideal power consumption at load level x by $PC_{ideal}(x) = x$.

With these definitions, we can express the energy proportionality $EP(x)$ of a system as function of the load level x:

$$EP(x) = \frac{PC_{ideal}(x)}{PC_{act}(x)} = \frac{x}{PC_{act}(x)} \tag{3.1}$$

This formula delivers EP values ranging from 0 to 1. Note, for $x < 1$ in a real system, $PC_{act}(x) > x$. According to our definition, each system

[1] By multiplying x by 100%, the percentage of system utilization can be obtained.

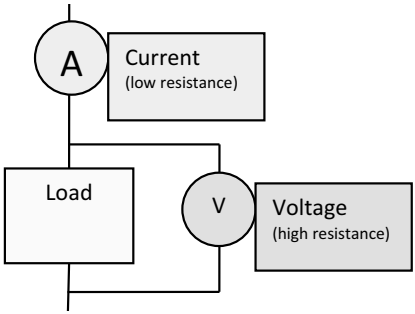


Figure 3.5: Measuring voltage and current

is perfectly energy proportional at $x = 1$. If a system reaches $EP(x) = 1$, it is perfectly energy proportional for all load levels x. In turn, the more $EP(x)$ deviates from 1, the more it loses its ideal characteristics.

Obviously, assessing energy proportionality is a lot more expressive than mere energy consumption. While the latter only captures a single point of the system's energy characteristics, the former reveals the ability of the system to adapt the power consumption to the current load.

3.6 Measuring Power & Energy Consumption

In order to measure energy consumption, both voltage and current need to be measured over time. Especially, alternating current (AC) requires detailed measurements, with a sampling rate higher than twice the frequency (50 Hz)[2], hence more than 100 Hz.

Figure 3.5 depicts a system under test (labeled *Load*), whose energy consumption shall get measured with attached voltage and current measurements. As the figure shows, voltage measurements are performed in parallel to the load, while current is measured in series. Hence, both measurements distort each other when performed simultaneously.

[2]Known as the *Nyquist–Shannon sampling theorem* [Sha49]

3.6.1 Voltage

Measuring voltage—more precisely, the *voltage drop* on the system under test (short SUT)—must be done in parallel to the load. The internal resistance of the instrument must ideally be infinite to not induce energy consumption and temper the current measurement. In practice, the internal resistance of the voltage measurement instrument is sufficiently high, i. e. $> 1M\Omega$.

3.6.2 Current

Measuring the current flow trought the SUT must be done in series to the load. Hence, the internal resistance of the instrument must be as small as possible to prevent voltage drops, which affect the measurement. Other, non-intrusive instruments are wrapped around the live wire, measuring the flowing current by secondary effects. Consumer grade current measurement devices use the *Hall effect*, i. e., a voltage difference in a conductor placed perpendicularly into the magnetic field of another current, relative to the intensity of the current, to calculate the current flow [Hal79]. An example of such instruments are *Current Transducers*, a combination of a *Current Transformer* with a *Converter* [Gui77]. The first transforms the current to a defined scale, while the second component converts the AC flow to a DC potential difference (voltage). The whole measurement is contact-less and its influence on the measurement is negligible.

3.6.3 Power

Since power is a product of voltage and current, both must be measured with sufficient accuracy to keep the total measurement error as small as possible. Special care must be taken for alternating currents, to differentiate between apparent and effective power.

More details about how to measure voltage, current, and power as well as descriptions of the instruments' internals can be found in [BEO10].

3.7 Measurement Device

In order to optimize and improve energy efficiency of a system, the first step is to exactly measure its energy consumption. An autonomous measurement track, allowing for long-running experiments and automatically correlating performance benchmark results with energy figures is necessary for repeatability of all experiments and to eliminate human error factors in measurements.

3.7.1 Problem description

A fast and accurate way to measure the power and energy consumption of servers and their components is required. For server components, individual power consumption should be assignable; hence, separate measures are needed for mainboard and storage disks. Further division in CPU and memory power consumption is desirable.

The measurements must be available digitally, to enable easy computer-aided processing and analysis. Moreover, the measurements should also include performance data of the SUT, e. g., CPU utilization, memory consumption, and disk and network load. It should be possible to relate measured data with performance measurements, either by annotating the data with exact timestamps or—preferably—by directly correlating energy and performance data. Since measurements are running over extended periods of time, up to several days, lots of measurements must be stored, while being able to aggregate and analyze them.

In summary, the measurement device should meet the following requirements:

Fast Data acquisition needs to be fast, i. e., done with high frequency to capture quick changes in power consumption.

Accurate The measurement error should be kept below 3% to be able to detect even small differences in power consumption.

Digital All measurements must be digitally recorded to automatically process, analyze, and summarize them.

Live Measurements should be instantaneously available for live monitoring or debugging.

Detailed Individual power rails should be independently and simultaneously measurable to enable measurements of different components separately.

Longevity Measurement must run for extended periods of time (hours to days), while the captured values need to be logged in high resolution for later analysis.

Correlated Data of different power lines and different types of measurements (power, performance) should be immediately correlated to simplify later analysis.

Compatibility The measurement device should support different systems-under-test, i. e., different hardware or components.

Portability The measurement device should support different analysis platforms, i. e., operating systems and software tools, to be able to do the analysis with various tools and helpers.

3.7.2 Existing measurement solutions

There exist various appliances to measure energy consumption of computers. In the following, well-known approaches are evaluated for their use in continuously monitoring servers and components as laid out previously.

Energy Meters For home users interested in power consumption of their devices, *Energy Meters* are available. These meters are plugged into the wall socket between the socket and the device and display current power and historic energy consumption on an LCD display. Some models also provide logging capabilities using storage cards.

While these meters are easy to use, they only provide coarse-grained measurements for all consumers plugged into the socket. Power consumption of individual components cannot be assessed separately. Accuracy of these devices is often questionable [Sti09].

Energy meters do not provide high-frequency measurements, typically, they show update rates of 1 Hz or below. The use case for these devices is short to mid-term evaluation of power consumers to evaluate possible energy savings. Therefore, typical measurement durations are

between hours up to weeks. Since the devices are already connected to power, they take their operating power out of the measured wires. Automated live monitoring is possible, whereas live correlation with other data is not viable, because measurement data have to be manually extracted from an attached storage card. Energy meters are compatible with all kinds of computers and devices, as long as they have a wall socket connector. Since all data extraction has to be done manually, the setup is portable across different analysis platforms, but requires lots of manual intervention.

Specialized meters The interest in measuring energy consumption arrived in datacenters as well. Therefore, *specialized energy meters* were developed to better suit the needs of datacenter operators. While the working principle stays the same as for traditional, multi-purpose energy meters, they offer various improvements to support their deployment in datacenters. First, a single meter is able to measure multiple connected servers, thus reducing the need of acquiring a multitude of energy meters and reducing space consumption in the server racks. Second, the measurements are available online, to allow integration in existing datacenter management solutions. Typically, they offer network interfaces (RJ45) to integrate directly into the datacenter's management network. Lastly, they provide additional features as monitoring the load balance in three-phase four-wire installations and alarm functions.[3]

While specialized meters offer some advantages compared to consumer energy meters, they still only measure the power consumption of entire servers. Measurements at component level are not intended.

The target audience for these meters are datacenter administrators, who are usually not interested in high-frequency power consumption data. Typically, hour-to-hour measurements and greater intervals are of interest here. Therefore, these meters still measure at rather low frequencies of 1 Hz or above.

While these specialized meters provide interfaces and APIs to integrate their readings into other monitoring solutions, portability comes with the high price of software customization. There is a broad range of specialized meters, each with different features. Therefore, a general statement regarding portability, compatibility, and other features is im-

[3] `http://www.powerleiste.de/smart-power-monitor.html`

possible to make. Yet, with these devices, the price usually determines the feature set, and suitable solutions are very expensive.

Current Clamps It is possible to manually measure the current flow of a single rail using a *current clamp* and a connected *multimeter*. The current clamp is wrapped around the electrical wire without physical contact. By measuring electromagnetic fields of the conductor (*Hall effect*), the flowing current can be estimated with sufficient precision (see below). The change in the electromagnetic field induces a voltage in the current clamps, which can be read out by an attached multimeter. The total measurement error is a combination of the current clamp's error and the multimeter's error.

While this combination makes it possible to measure with sufficient accuracy, standard multimeters only display the measured values but do not log them. More expensive logging multimeters are needed, which digitally save the measurements on a storage card[4]. Hence, live logging is possible, but measurements must be correlated manually with performance data. Detailed logging of additional rails requires the same setup multiple times, with no automatic correlation between the measurements. In theory, the measurement duration is only limited by the storage space for log data and the battery capacity of the devices. According to our experience, current clamps quickly drain batteries, especially with continuous measurements. Therefore, measurement duration is usually limited to a few hours.

This setup is compatible with almost all hardware and components, as long as the power rails can be clamped into the measurement device. Since all data extraction has to be done manually, the setup is portable across different analysis platforms.

Another option in connection with current clamps is the use of digital oscilloscopes for displaying and logging data. High-end oscilloscopes offer data logging and also interfaces to connect to computers for data analysis, e. g., over USB. Yet, the setup is complex.

In summary, using current clamps to measure power consumption is viable only for one-time, non-continuous, manual measurements. This setup is not qualified for continuously monitoring power consumption of servers or components.

[4] `http://www.fluke.com`

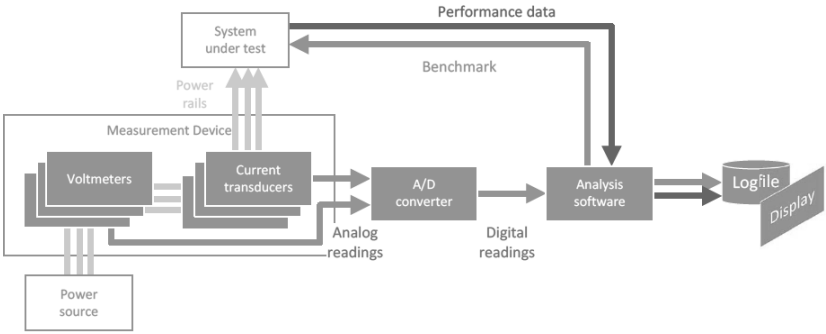


Figure 3.6: Measurement track and required peripherals

Custom power meter Since none off-the-shelf available solutions did satisfy the specified requirements, a custom development was needed. As previously described, current transducers are able to quickly and accurately measure current and can easily packed in groups to measure multiple rails simultaneously. To digitize the output voltages, an A/D converter is needed.

To satisfy all requirements listed above, we decided to develop the whole measurement track ourselves, starting with measuring the current and voltages of the SUT, converting them to digital readings, and analyzing and storing them using custom software. Figure 3.6 sketches a draft of the measuring track. On top, the system under test is shown. System under test can be a single computer, measuring each component individually, or a cluster of servers, measuring power consumption of each node. To track power consumption, current transducers and voltmeters are intercepting the electric wires powering the SUT. An A/D converter polls the instruments' readings and makes them digitally available to a connected analysis computer. The analysis computer, running the benchmark suite described in Section 4.5, also receives performance readings from the SUT and correlates both sources. For immediate analysis, the measurements can be shown on a display. Additionally, all results are logged to disk for later evaluation.

Pin		*Voltage*	$Current_{max}$
ATX Connector			
1		3.3 V	19 A
2		5 V	19 A
3		12 V	20 A
4		3.3 V	2.5 A
5		-12 V	0.3 A
HDD Connector			
6		12 V	2 A
7		5 V	2 A
SATA Connector			
8		3.3 V	2 A
9		5 V	2 A
10		12 V	2 A

Table 3.1: Points of measurement to determine component-wise power consumption of a single machine

3.7.3 Single-server measurement device

A first implementation of the analysis software and the measurement device was introduced 2009 in [Sch09]. This initial version was designed to monitor performance and energy consumption of XTC, a native XML DBMS[5]. To assess power consumption of various components, 10 current transducers were used. Table 3.1 lists all measurement points monitored by the device. Each point represents a power rail with a specific voltage. To measure power consumption of a rail, it is fed through a current transducer.

With this first device, power consumption of the system board and two attached disks can be measured in parallel. All measurements were read by an A/D-converter (Labjack UE9 [Lab08]) and transferred to a connected monitoring PC, running a Java-based analysis software.

[5] `http://xtc-project.de`

	Voltage	$Current_{max}$	$Power_{max}$	**Comment**
1 x	230 VAC	50 A	2,500 VA	Apparent Power
1 x	230 VAC	-	-	Voltage
12 x	230 VAC	5 A	250 W	
		- or -		
10 x	230 VAC	5 A	250 W	
2 x	230 VAC	10 A	500 W	

Table 3.2: Points of measurement to determine server-wise power consumption in a cluster of nodes

3.7.4 Cluster measurement device

To extend the monitoring capabilities to a cluster of nodes, a new measurement device was needed. Instead of monitoring individual power rails of a single computer, the overall power consumption of up to 10 nodes can be measured with this device. Further, the supply voltage (230 V AC) and the apparent power are available. The device outputs all readings as analog voltages between 0 and 5 volts, to be able to be read by a connected analog/digital converter. The remaining part of the monitoring infrastructure did not need to be changed.

Table 3.2 lists the measurement points with their maximum power range. To support a wider range of hardware, one of the plug-in-modules was enhanced to support loads of up to 500 Watts. All readings are translated into output voltages between 0 and 5 volts—the input range of the connected analog/digital converter.

The measurement device, built by the electronics workshop of the University of Kaiserslautern, is modularly structured: A *backplane* is connecting all components and distributes supply voltages inside the device. Connected to the plane, 7 *plug-in modules* are connected; 6 for power measurement of individual nodes and one to measure apparent power and the AC voltage.

Figure 3.7 sketches the plug-in module power measurements. On each module, this circuit was repeated twice to be able to measure two consumers with each card. On the left, the two inputs are illustrated: Supply voltage and the 230V rail under measurement. The current transducer measures the current flow, which needs to be mul-

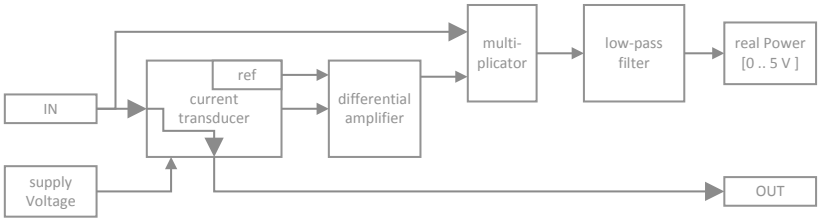


Figure 3.7: Plug-in module for measuring power consumption

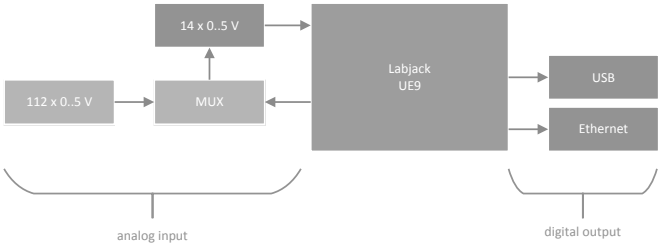


Figure 3.8: Labjack overview

tiplied with the instantaneous voltage to get the instantaneous power consumption of the connected system. On the right, the out-connector for the component-under-test and the connector for reading the measured values are shown.

A more detailed explanation of the measurement device, including circuit diagrams, can be found in [Rit11].

Labjack Labjack is a data acquisition device (DAQ), which we used as an analog/digital converter (ADC) to convert voltage/current/power consumption readings to digital data. The used Labjack UE9 device[6] can read the voltage of up to 14 discrete analog input ports and supports additional multiplexers to split each input into 8 additional channels. Labjack measures voltages between 0 and 5 V with 12 to 16 bit accuracy. Dependent on the number of channels as well as the resolution and multiplexing used, measurement frequency can go up to 1,000 Hz.

[6] `http://labjack.com/ue9`

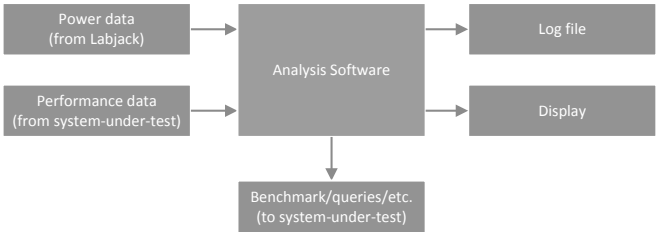


Figure 3.9: Software overview

Labjack also supports output channels, digital I/O and timers, which was unused in our measurements. Readings can either be queried by USB or Ethernet in streaming mode, i. e., Labjack will send a continuous stream of measurements to the requester. In Figure 3.8, an overview about Labjack's mode of operation is given.

Analysis software After converting the analog values to digital readings, an analysis software can read them from Labjack and correlate the power measurements with performance data, as sketched in Figure 3.9. The software, written in C#, is also hosting a benchmark component, able to send predefined sequences of queries to the SUT, and correlating each step in the benchmark with respective performance and power consumption figures.

3.7.5 Accuracy estimation

To estimate the measurement error, the error of the whole measurement track needs to be assessed and compared to a reference measurement. The resulting error accuracy cannot exceed the reference accuracy; therefore, a calibrated device with sufficient precision must be used. For the following measurements, a *Fluke 702 Calibrator* was used as reference point. This device is calibrated to a measurement error below 0.25% in the operating range [Flu94].

First, we have estimated the error for the Labjack ADC and the measurement device separately. Since the software component running on the analysis computer processes the raw input values from the Labjack, we did not estimate accuracy for this component.

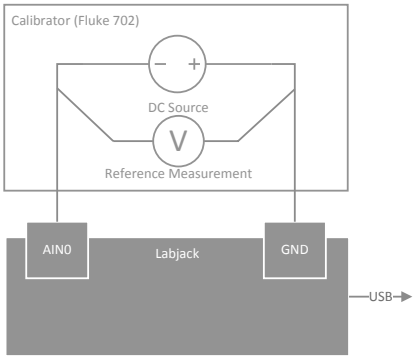


Figure 3.10: Setup for accuracy estimation of the Labjack UE9

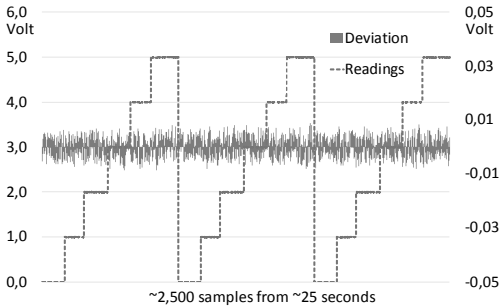


Figure 3.11: Results from Labjack UE9's accuracy estimation

Labjack To estimate the measurement error of the Labjack ACD, we have connected the calibrator to one of Labjack's analog input channels (AIN0) as depicted in Figure 3.10. The calibrator was set to output fixed voltages between 0 and 5 V. By comparing the voltages set and Labjack's readings, we can directly see the measurement error of the DAQ. In Figure 3.11, the results of the measurements are visualized. The measurements were performed using Labjack's streaming mode, sampling the input at 100 Hz; hence, the entire measurement took about 25 seconds and resulted in 2,500 samples. The red line denotes the values read from Labjack (primary Y-axis), while the blue bars depict the deviation to the Fluke 702 reference voltages on the secondary Y-axis.

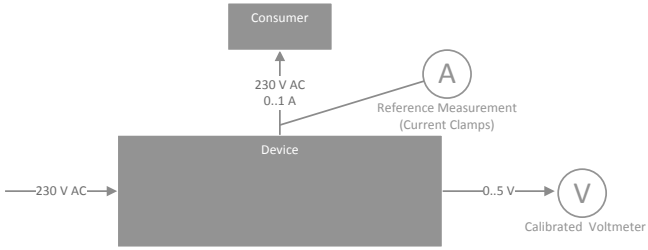


Figure 3.12: Accuracy estimation of the measurement device

For simplicity, the reference voltages are not shown, as they are more or less identical to Labjack's readings. As the results show, Labjack measures the supplied voltages with high accuracy, where the error is less than ± 0.01 volt.

Measurement Device To analyze the error of the measurement device, we compared its output with the measurements of a reference current clamp as depicted in Figure 3.12. The measurement device provides "maintenance ports" to access the output voltages sent to Labjack with a voltmeter directly. We used the Fluke 702 Calibrator again to read the output voltages from the device with high accuracy. Yet, the current clamp used (an *ESI #695 80A Current Probe*) is not calibrated and has an unknown error margin of 3% and more [Ele03]. Error estimation of the measurement device is therefore tainted with high uncertainty.

Since we did not have any load generators available to simulate electric loads between 0 and 200 watts, we have connected various electric consumers to the device, providing static power consumption, e. g., a series of 40 W light bulbs. Figure 3.13 compares readings of the measurement device and the current clamps. The X-axis denotes the reading from the clamps in watts, while the bars plot the absolute deviation of the measurement device. The absolute error varies between 0.1 and 3 watts, resulting in a relative error of 3% in average. This error margin is identical to the accuracy of the clamps. Therefore, using this experiment, we are unable to tell whether the device has an even lower error margin. Yet, an error of 3% would be acceptable for the whole measurement track.

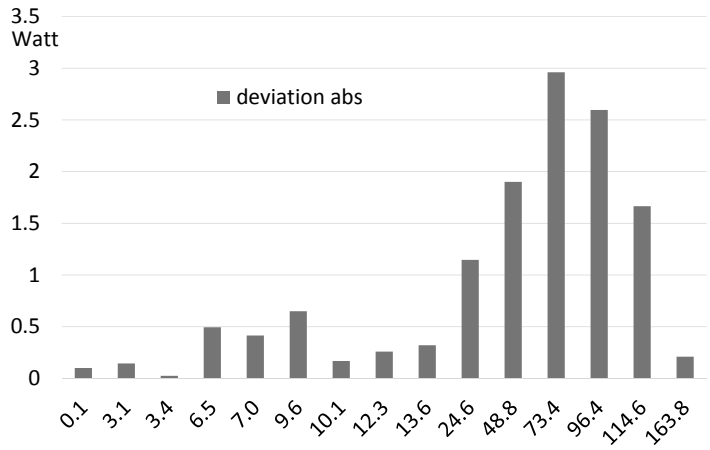


Figure 3.13: Results from the device's accuracy estimation

Software The analysis software running on a dedicated computer reads the values from the Labjack device via USB and translates the raw voltage readings into power consumption (watts). This is done by fixed factor multiplication. To measure energy consumption over time, a pseudo-integral is formed by summarizing the product of the power consumption with elapsed time since the last measurement. A (theoretical) rounding error while processing the digital readings seems possible, but would be far too small to measure. Therefore, we did not conduct a separate accuracy estimation for the software components.

Measurement track Finally, to estimate the error of the whole setup, we assembled the measurement track and started to collect power consumption values. Again, we connected various electrical consumers to the track and compared the track's readings to reference readings from parallel attached current clamps, depicted in Figure 3.14. Since the current clamps, as explained earlier, are not very accurate themselves, we have added an energy cost measuring device (Voltcraft Energy Logger 3500) to track energy consumption directly at the power outlet. According to the documentation, this device has a measurement error below 1% for loads from 5 to 3.500 watts [Vol08]. Unfortunately, be-

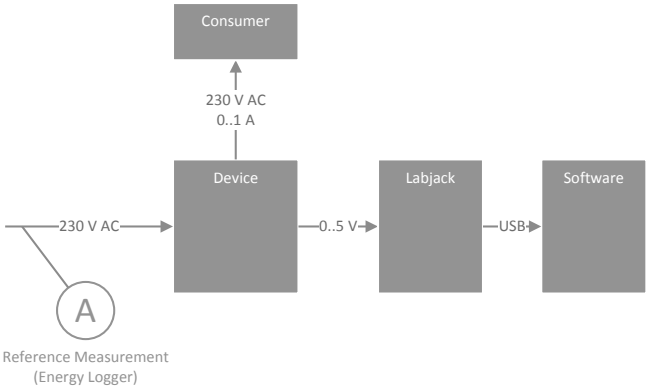


Figure 3.14: Accuracy estimation of the measurement track

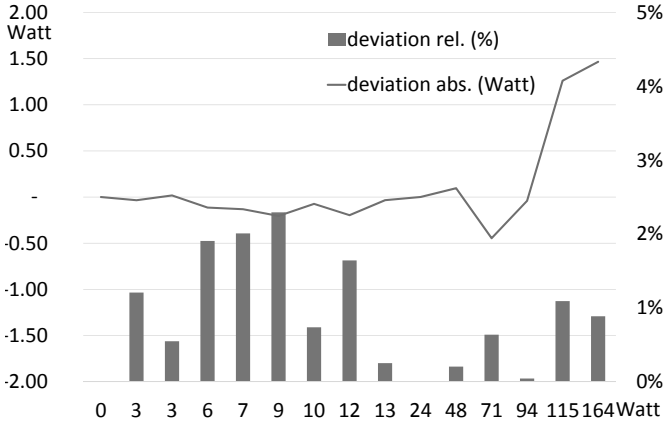


Figure 3.15: Results from the measurement track's accuracy estimation

time	Energy Logger	Measurement Track		deviation	
[s]	[Wh]	[Wh]		[Wh]	[%]
7628	7.16	7.54		0.38	0.89
7762	26.30	27.03		0.73	1.17
7607	99.28	101.89		2.61	1.93
7498	195.23	199.88		4.65	2.02
7666	335.61	336.52		0.91	0.24

Table 3.3: Long-running accuracy estimation measurement

low 5 W, accuracy drops to ± 5%. Since the measurement device itself consumes power, we have to adjust all readings from the energy logger by subtracting 16.8 watts (determined in previous experiments). In the following paragraphs, these corrections have been made without explicit notice.

In Figure 3.15, we have compared the readings of the logger to the readings of the other two devices. The X-axis plots the readings from the logger, while the line graph shows absolute deviation of the measurement track from these readings (primary Y-axis). The solid bars depict relative deviation on the secondary Y-axis.

As the results indicate, the measurement track exhibits a maximum deviation of 2.5%. Mostly, measurement error is below 1%, compared to the energy logger.

Long-time measurement With the energy logger, we are able to conduct long-running energy consumption analysis of connected devices. We have set up a long-time measurement, running for approximately 2 hours to estimate the measurement bias in a longer running experiment. The measurement set-up is the same as in the previous experiment. Again, some electric consumers are connected to the measurement track. The power cord is plugged into the energy logger. We turned on the energy logger and the measurement track to continuously read the power consumption (watts) and automatically integrate the readings to output energy consumption (watt hours).

In Table 3.3, the results of the long-running measurement are shown. Figure 3.16 visualizes these results. We have connected various devices, steadily consuming between 3 and watts. After two hours, we compared

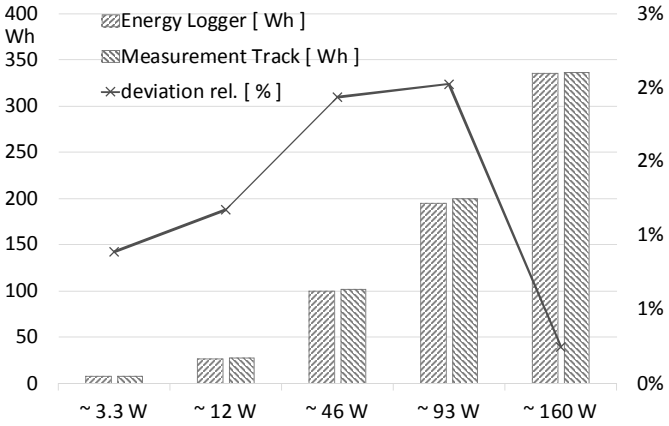


Figure 3.16: Long-time accuracy estimation

the readings from the logger with the measurements of our track. As the results show, our measurement track deviates from the energy logger by less than 3%, which is perfectly acceptable. Especially the long-term abberation in the typical working region of a single server—between 20 and 40 watts—is with less than 2% perfectly fine.

4 Benchmarking for Energy Efficiency

In the past, benchmarking (database) servers was primarily focused on performance. Since database machines are tailored to high query throughput and fast response times, traditional benchmarks rank systems by these two measures.

With growing energy concerns, the view has broadened to energy-related aspects as well. Recently, well-known benchmark standardization committees like *TPC* and *SPEC* have included energy aspects in their benchmarks.

Because electricity prices are rising and operating costs draw close to acquisition cost [PN08], performance per watt as an additional indicator gained growing interest in recent years.

4.1 Related Work

Before introducing new benchmarking paradigms, existing benchmarks are presented. First, this work lists classical benchmarks, with focus on performance and costs (not energy costs). Next, more recent proposals and extensions are presented, which include an energy-related component.

4.1.1 TPC-C

The *Transaction Processing Performance Council* (TPC)[1] is a non-profit organization with the "objective to define transaction processing and database benchmarks and to disseminate objective, verifiable TPC performance data to the industry" [TPC10a]. TPC is composed of

[1] `http://tpc.org/`

members of networking and database companies, e. g., Cisco Systems, HP, IBM, Microsoft, and Oracle (as of January 2014).

TPC-C is an On-Line Transaction Processing (OLTP) benchmark, proposed by TPC in 1992. It simulates an environment with a number of users submitting concurrent queries against a database. The database is simulating an on-line warehouse, enabling the users to query the stock, enter and deliver orders, update payment information, and check the orders' status.

Performance is measured in new-order transactions per minute. The transaction rate (tpmC) is expressing the performance of the system, whereas the associated price per transaction ($/tpmC) denotes the price of the SUT in relation to its performance.

A representative system, as of 2011, running the TPC-C benchmark consists of multiple nodes, each equipped with multiple CPUs and up to 512 GB of DRAM. Additionally, the persistent storage is consisting of several thousand disk drives [SHK12].

4.1.2 TPC-E

TPC-E is another On-Line Transaction Processing (OLTP) benchmark, proposed by TPC in 2007. In contrast to TPC-C, this benchmark simulates an on-line brokerage firm [TPC10b]. Customers place orders, query accounts, and perform market research by submitting queries to the brokerage database. The broker submits queries to the market and updates order information. Customers and market are part of the test driver, while the brokerage firm—and the containing database—represents the SUT.

Performance is measured in transactions per second (tps), specifically, in *Trade-Result* transactions per second.

A single-server-based system placed in the top ten results of the TPC-E benchmark (as of 2011) requires about 4,500 watts of electricity to supply its hardware, consisting of 8 multi-core processors, about 800 hard disk drives, and 2 TB of DRAM [SHK12] .

4.1.3 TPC-H

Besides OLTP benchmarks, TPC has also standardized On-Line Analytical Processing (OTLP) benchmarks. Currently in use is TPC-H,

an OLTP benchmark proposed in 1999 [TPC13]. It is designed as a decision-support benchmark, simulating ad-hoc queries with concurrent data modification on large volumes of data. Compared to OLTP, the queries are more complex and involve large amounts of data, simulating typical data-mining business questions. The size of the warehouse (scaling factor) and the number of parallel clients (called streams), submitting queries, determine the database utilization.

TPC-H measures performance using a complex metric called the *TPC-H Composite Query-per-Hour Performance Metric* (QphH@Size). It includes several aspects of the system, e. g., database size and query throughput using a single client and throughput using multiple concurrent clients. Additionally, the price (acquisition + maintenance cost over a fixed period of time) is included in the TPC-H Price/Performance metric, expressed as $/QphH@Size.

4.1.4 TPC-Energy

The three previously presented benchmarks do not include electricity cost in their metrics. Although the benchmarks mention maintenance costs for a typical usage period (5 years), they do not include energy costs for that duration.

TPC noticed the rising demand for energy-related figures and released a benchmark add-on in Dec. 2007, called *TPC-Energy* [TPC10c]. This benchmark is an enhancement of the existing TPC-C, TPC-H, and TPC-E benchmarks. It introduces energy measurement specifications and metrics to extend all three benchmarks.

TPC-Energy's primary metric is *watts per performance*, with performance defined by the respective underlying benchmarks, e. g., *tps* for TPC-E or QphH@Size for TPC-H. The metric is defined using the total energy consumed by the SUT and dividing it by the normalized work performed in the same measurement interval.

Supplemental, secondary metrics are defined to further characterize the system's energy consumption: Power consumption of the system can be broken down to assess power/performance metrics for subsystems, e. g., storage system or application layer. Further, the idle power consumption of the SUT is reported.

In addition to these metrics, the benchmark introduces a software package helping to facilitate the implemention of the TPC-Energy

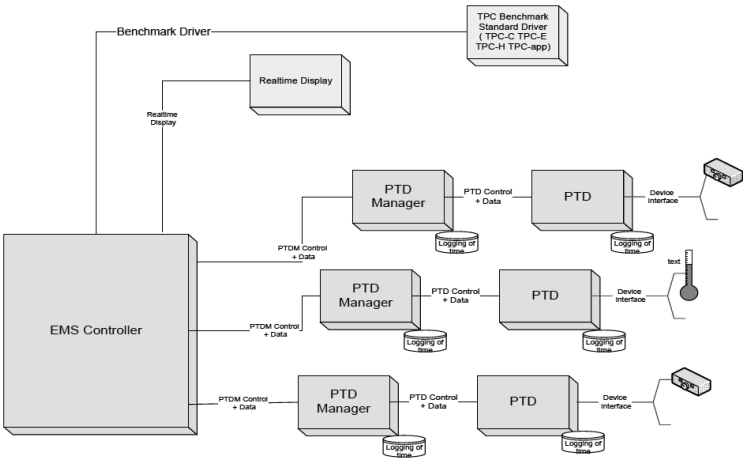


Figure 4.1: TPC-Energy Measurement System, from [TPC10c]

specifications, called *TPC-Energy Measurement System* (EMS). EMS is providing power instrumentation interfaces and logging facilities for power and temperature. Furthermore, it enables easy report generation. Figure 4.1 depicts the experimental setup for TPC-Energy benchmarks, including the EMS Controller for correlating readings with benchmark steps and a number of *Power and Temperature Daemons* (PTD), that report power consumption and temperature of the connected part of the SUT to the controller.

4.1.5 SPECpower_ssj2008

The *Standard Performance Evaluation Corporation* (SPEC), founded in 1988, is a consortium of over 60 hard- and software manufacturers with the aim to define standardized, realistic performance tests [Sta95]. SPEC has published a broad variety of benchmarks for different kinds of SUTs, e. g., *SPECjvm2008* for measuring the performance of a Java Runtime Environment (JRE) on different platforms, *SPEC CPU2006* for evaluating CPU performance with compute-intensive workloads, and *SPECsfs2008* for benchmarking speed and throughput of network file system servers.

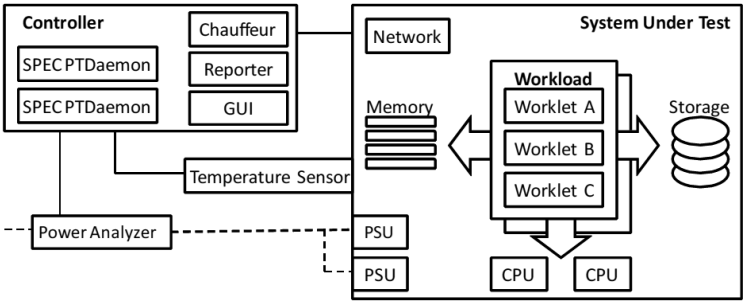


Figure 4.2: SERT design overview, from [Sta13]

In December 2007, SPEC released a new benchmark measuring system performance and power consumption, called *SPECpower_ssj2008*. Unlike previous energy benchmarks, SPECpower_ssj2008 measures performance and power at various utilization levels between idle and 100% load. Hence, this benchmark does not only report power consumption for performance-oriented workloads, but also identifies energy characteristics of systems under variable utilization. To simulate a non-peak load, the benchmark driver introduces random pauses between sending requests to the SUT. For example, to simulate a 30% utilization, the benchmark pauses for an average of 70% of the time during the measurement interval. Hence, after the measurement period, the SUT was only stressed with 30% of the requests, compared to peak utilization. After measuring 11 load levels from idle to 100% in steps of 10%, the geometric mean of them is used to calculate a single result, denoting the overall energy efficiency of the SUT.

SPECpower_ssj2008 is Java-based and therefore compatible with a broad range of systems and operating systems.

4.1.6 Server efficiency rating tool

SPEC has also published a tool called *Server Efficiency Rating Tool* (SERT), which can be used—similar to TPC-Energy—to estimate a system's power and energy consumption. While previous SPEC benchmarks focused on isolated components and performance indicators, the design rationale behind SERT was to rate the overall efficiency of all

kinds of servers, from centralized systems to distributed clusters. Therefore, a highly scalable workload is needed to adjust the benchmark to the size of the SUT. Like all other of SPEC's benchmarks, SERT is Java-based, running micro-benchmarks like sorting, encryption, compression, and hashing.

Figure 4.2 sketches SERT's system design. Similar to TPC-Energy, so-called "Power and Temperature Daemons" (PTDs) monitor power consumption and temperature of all parts of the SUT. A controller is coordinating the benchmark and collecting all readings from PTDs.

Workloads in SERT consist of warmup, measurement and post-measurement phases, in alternation with idle phases. Therefore, a system's ability to quickly power down in idle times is an important aspect of its overall energy efficiency.

Each of the micro-benchmarks grades the system in three metrics: Power consumption, performance and efficiency. While power consumption is depicting the overall wattage of the SUT, performance and efficiency are dimensionless score points.

4.2 JouleSort

A common task in all software, especially databases, is to sort data sets. Typical benchmarks focused on sorting do only report performance, hence, how fast a data set can get sorted, using various algorithms on different hardware. The idea of *JouleSort* is to benchmark, how much data a SUT can sort in relation to it's energy consumption [Riv+07]. Hence, instead of the metric *sorted records per second*, this benchmark has chosen *sorted records per joule*.

Unfortunately, JouleSort is only focused on peak-load sorting, hence, at full system utilization. The workload is also very limited (sorting only). Yet, JouleSort is one of the first benchmarks on energy efficiency.

4.3 Benchmark Requirements

Based on previous observations about typical database server utilization, existing benchmarks do not represent realistic workloads. This is not an issue when measuring performance, since only peak performance

is of interest, but when it comes to energy, a more detailed look at the workloads is necessary to be able to reproduce real-world utilization in a benchmarking environment and to truly estimate energy consumption of a system.

In order to report detailed energy characteristics of a system and to compare them to others, a new benchmarking paradigm is needed. With TPC-Energy and SPECpower_ssj2008, first steps have been taken in assessing energy consumption while benchmarking, yet both specifications have shortcomings.

TPC-Energy measures power consumption only in conjunction with performance benchmarks and at idle. Hence, it only covers two single points of utilization of the SUT.

SPECpower_ssj2008 defines 10% intervals of utilization, where power consumption is measured. Although SPEC presented a more detailed specification than any energy benchmark before, it is still measuring a limited subset of the whole utilization spectrum under static workloads.

Since all benchmarks rely on measurements of static workloads, with no variance in utilization over time, they do not reflect the dynamics of the SUT and its ability to adapt to workload changes. Traditional servers did not implement sophisticated energy-saving mechanisms and, therefore, did not exhibit prolonged setup times to change from near-idle to full processing power. Therefore, existing benchmarks were adequate for those systems.

With the emergence of new technologies and more advanced energy-saving approaches, systems do need additional time to change between their usage states. Hence, existing benchmarks do not model adequate usage patterns with dynamically changing workloads and are therefore unable to estimate energy consumption and performance characteristics of these new systems in realistic scenarios.

4.4 Benchmark Proposal

Based on the observations in the previous section, it is easy to see that current server installations do not behave like the systems measured in traditional benchmarks. While benchmarks usually measure peak performance, typical servers operate far away from that point during most of the time. Nevertheless, benchmark results are comparable and mean-

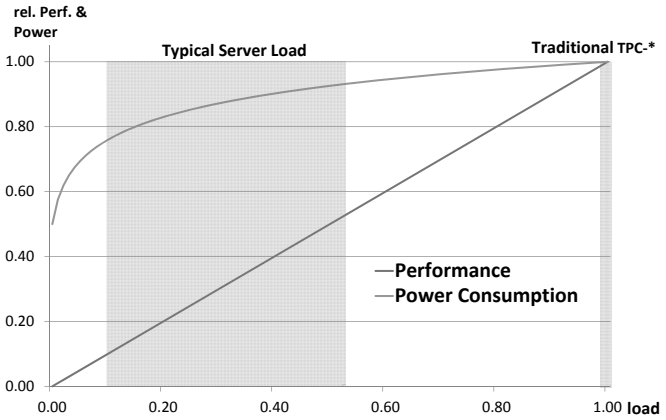


Figure 4.3: TPC-* vs. the real world, from [SHK12]

ingful when it comes to performance only. As long as attention is not turned to energy consumption, the mismatch between benchmarking and real usage of servers does not carry weight. Performance measurements under peak utilization can be easily broken down to lower load situations. Hence, a system, able to process *x tps per second* at peak, can also process *0.5x tps per second*.

In contrast, energy-related measurements obtained at some point of utilization are not transferable to other load situations because of the non-linear scaling of energy consumption of today's computer hardware. Therefore, the whole span of different load situations should be measured separately to obtain meaningful energy data for customers.

As an analogy from a well-known field, automobiles are benchmarked similarly with additional "energy-related" measures. Hence, the power of a car is estimated by its horse power and its top speed, like database servers are classified by their hardware and their peak *tpmC / QphH / tpsE*. On the other hand, the gas consumption of a car, estimated at top speed, is meaningless for the average driver, because the measurement does not indicate the gas consumption for average usages. Therefore, a car's mileage is measured by driving the car through a set of standardized usage profiles which reflect the typical use of the vehicle. The same paradigm should be applied to database benchmarks as well, where en-

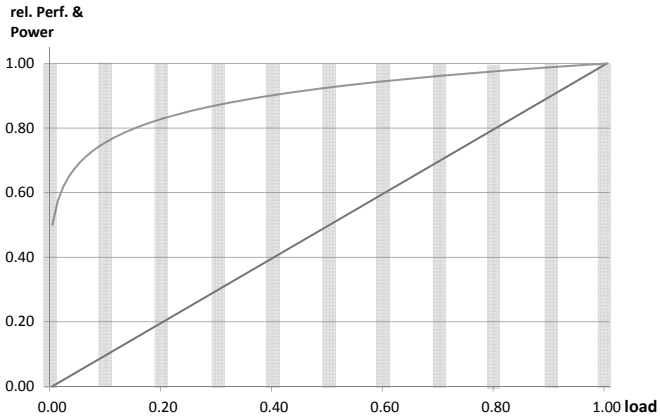


Figure 4.4: Static energy-efficiency measurement, from [SHK12]

ergy consumption measured at peak utilization is no indicator for the average use case.

Figure 4.3 depicts performance and energy efficiency of an exemplary server. The performance-oriented TPC-* benchmarks cover the light-blue region of the server's power/performance spectrum, while the typical working region—colored in light-red—is not examined. The performance difference is not problematic, but there exists a large discrepancy in energy efficiency between both regions. With the TPC-(C/E/H) benchmarks, even in conjunction with TPC-Energy, energy efficiency of the typical working region of a server is not analyzed.

To cope with the limitations we have outlined previously and to keep the TPC benchmarking suite up to date, we propose a new paradigm in benchmarking.

4.4.1 Static weighted energy proportionality

Nowadays, the measurement paradigm for the TPC benchmarks strictly focuses on performance results, i.e., to get the most (in terms of units of work) out of the SUT. Hence, this methodology collides with the desire to get a meaningful energy-efficiency metrics for the system. Therefore, we propose a sequence of small benchmarks that utilize the SUT at

different load levels, instead of a single run at peak load. Figure 4.4 depicts a feasible set of benchmark runs at different utilization ratios. First, a traditional TPC-* run will be performed, i.e., at full utilization. That run is used as a baseline to get the maximum possible performance the SUT can handle. Next, based on the results from the first run, the number of queries per second issued for the other runs is calculated using the follwing equation, where x denotes the system utilization between 0 and 100%:

$$\begin{aligned} baseline &:= \frac{transactions}{second} \text{ @100\%} \\ \frac{transactions}{second} \text{ @}x &:= baseline \cdot x \end{aligned}$$

Depending on the type of benchmark, the characteristics and knobs for throttling can differ, e.g., for TPC-C increasing the think time seems reasonable while for TPC-H a reduction of concurrent streams is the only meaningful option. We call this a *static weighted energy-proportionality benchmark*, because the workload does not change in between and, therefore, the system does not have to adapt to new load situations. To allow the system adapting to the current workload, a preparation phase of a certain time span is preceding each run. During the preparation time, the SUT can identify the workload and adapt its configuration accordingly. It is up to the system, whether and how to adapt to the workload, e.g., the system can power down unused CPU cores or consolidate the workload on fewer nodes in order to save energy. After the preparation phase, the overall energy consumption during the run is measured. In other words, instead of measuring the performance of the SUT, we are now measuring the power consumption for a certain system usage.

At each load level, the system's energy proportionality (according to equation 3.1) is calculated. By multiplying each result with the relative amount of time the system is running at that load level, we can estimate overall energy proportionality for arbitrary workloads.

Formula

Let EP_i be the energy proportionality at load level i, and let T_i be the relative time, the system is at that level. Then the static weighted

(a) Typical, real-world system

load	rel. time	rel. PC	EP	EP · time rel.
idle	0.11	0.47	0.00	0.00
0.1	0.08	0.7	0.14	0.01
0.2	0.19	0.78	0.26	0.05
0.3	0.23	0.84	0.36	0.08
0.4	0.18	0.88	0.45	0.08
0.5	0.10	0.91	0.55	0.05
0.6	0.05	0.93	0.65	0.03
0.7	0.02	0.94	0.74	0.01
0.8	0.01	0.98	0.82	0.01
0.9	0.01	0.99	0.91	0.01
1.0	0.02	1.00	1.00	0.02
SWEP ($= \sum$)				0.37

(b) Energy-proportional system

load	rel. time	rel. PC	EP	EP · time rel.
idle	0.11	0.0	1.0	0.11
0.1	0.08	0.1	1.0	0.08
0.2	0.19	0.2	1.0	0.19
0.3	0.23	0.3	1.0	0.23
0.4	0.18	0.4	1.0	0.18
0.5	0.10	0.5	1.0	0.10
0.6	0.05	0.6	1.0	0.05
0.7	0.02	0.7	1.0	0.02
0.8	0.01	0.8	1.0	0.01
0.9	0.01	0.9	1.0	0.01
1.0	0.02	1.0	1.0	0.02
SWEP ($= \sum$)				1.00

Table 4.1: Example calculation of the SWEP

energy proportionality of the system (= $SWEP$) can be calculated as:

$$SWEP = \int_{i=0.0}^{1.0} EP_i \cdot T_i \; di$$

In this formula, relative time is used to denote the fraction of time in a certain load interval. By integrating, i. e., summing up, over all time intervals, the power consumption over the total runtime (100%) is calculated.

We can estimate the power consumption (= PC) of the SUT during the measured interval by multiplying the (absolute) power consumption in each interval (PC_i) with the relative time, the system is in that load interval:

$$PC = \int_{i=0.0}^{1.0} PC_i \cdot T_i \; di \qquad [watts]$$

Again, time is expressed relative to the total runtime, hence, the unit of this forumla is watts, not watt seconds.

Further, by adding the system's performance to the formula (denoted as *tps* in the following), we can estimate the overall energy efficiency.

$$EE = \int_{i=0.0}^{1.0} \frac{PC_i}{tps_i} \cdot T_i \; di \qquad \left[\frac{watts}{tps}\right]$$

In reality, the integral is approximated by the sum of load situations measured, e. g., by eleven measurements at loads from 0% to 100% at 10% increments.

This formula puts power consumption in relation to transaction throughput, and thus, enables energy efficiency comparison of different systems running the same workload. The higher the energy efficiency, the better a given system transforms electricity into work. This formula uses the same unit as the TPC-Energy benchmark, which also applies *power consumption over throughput* as its metrics. The rationale of

the TPC for choosing this metric was the desire to be similar to the traditional TPC metrics *price over throughput* and, furthermore, to allow a secondary metrics for each of the subsystems.

Example

To clarify the calculation of the weighted average, we will give an example using the load and energy measurements provided by Google (see Figures 2.3 and 2.4). Table 4.1 shows the average power consumption and time fractions of a hypothetical server for 11 utilization levels. The data is derived from the two studies done by Google. Additionally, a perfectly energy-proportional system is shown.

This static approach has certain drawbacks: First, the measurements are rather coarse-grained in practical applications, i.e., "reasonable" static measurements will be employed at 0, 10, 20, ..., 100% load, but not in greater detail. And second, this calculation does not take transition times from one load/power level to another into account.

4.4.2 Dynamic weighted energy efficiency

To design an energy-related benchmark that overcomes the drawbacks of the static approach, we are proposing a refinement of the previous benchmark, called *dynamic weighted energy-efficiency benchmark* (DWEE). In order to simulate an even more realistic workload on the SUT, the static measurements at various load levels of the SWEP benchmark are replaced by continuous sequences of different length and different load situations (so-called *scenes*), followed by each other without interruption or preparation times. In contrast to the static approach, all scenes run consecutively, thus transition times are measured as well using this benchmark. That enables us to test the system's ability to dynamically adapt (if possible) while running.

Every scene will run for a defined time span T, as sketched in Figure 4.5. A *time span* is always a cardinal multiple of a constant *time slice* t, thus, all scenes run for a multiple of that time slice.

The dynamic energy-efficiency benchmark should simulate a typical workload pattern, hence, the sequence of load levels should reflect the intended usage pattern of the SUT.

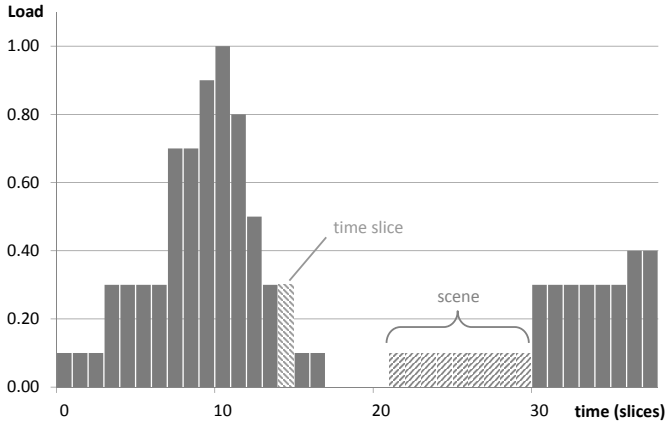


Figure 4.5: Dynamic weighted energy-efficiency benchmark – sample load pattern

Influence of the length of the time slice *t*

By adjusting the cardinal time slice t to smaller values, all benchmarking scenes will be shorter, hence, the system must react faster to changing loads. Such a course of action enables testing the SUT's ability to quickly react to changes. Of course, benchmark results can only be compared by choosing the same time slice t and the same sequence of scenes.

The minimum length of the time slice should not go below 10 minutes, because real-world utilization usually does not change faster than that.

Formula

Calculating the result of the DWEE benchmark is simpler than calculating the SWEP results, because the weighting of the utilization is determined by the selection of scenes. Since we are measuring the overall throughput and energy consumption, we do not have to aggregate several measurements. To obtain comparable benchmarks, the resulting measure of a run will be expressed in watts per tps. The result of the

dynamic weighted energy-efficiency benchmark, short *DWEE*, is:

$$\begin{aligned} DWEE &= \frac{overall\ Energy\ Consumption}{overall\ \#\ of\ Transactions} \\ &= \frac{joules}{transactions} \\ &= \frac{W}{tps} \end{aligned}$$

Hence, by employing the same sequence of scenes and the same length of t, the energy efficiency of different systems can be compared to each other. Since the benchmark simulates a daily workload, the average energy consumption of the SUT in a real-world scenario can be estimated.

4.4.3 Energy delay product

The product of energy consumption (joules) and runtime (seconds) is called *Energy Delay Product* (EDP), a metric originally stemming from chip and microprocessor design [GH96]. EDP is defined as follows:

$$EDP = \text{execution time} \times \text{energy consumption} \quad or$$
$$EDP = \text{execution time}^2 \times \text{power consumption}$$

The rationale behind this product is, that a system, running twice as fast and with twice the power consumption, exhibits the same EDP, hence, it is not better or worse than the original system. Another system, consuming the same amount of energy but delivering results faster will have a lower EDP. Likewise, lowering energy consumption while keeping runtimes unchanged results in a better EDP. Therefore, the Energy Delay Product is a good measure to compare the energy consumption/performance of heterogeneus systems, given the same task. The lower the EDP, the more energy efficent a system is.

Yet, EDP values are only comparable when measured using the same workload.

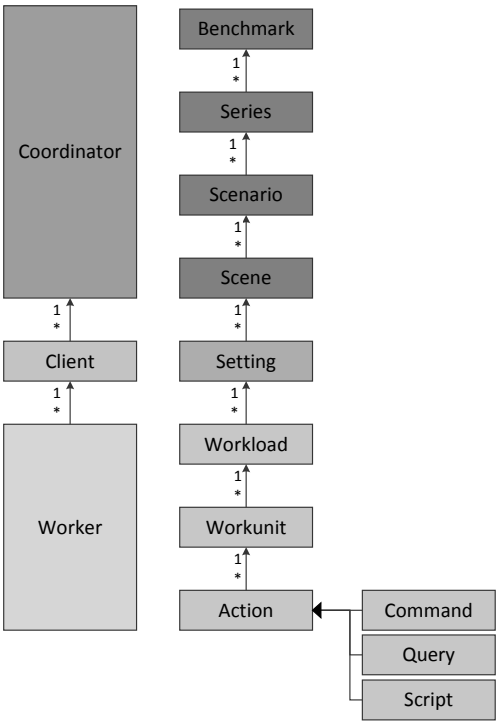


Figure 4.6: Architecture of the benchmark software

4.5 Benchmark Software

To run the proposed benchmark against our database cluster, we have developed a benchmark suite, based on an earlier implementation for *Brackit* [Bäc13]. The benchmark suite is XML-based and allows to define long series of benchmark runs, consisting of variable intensities to realistically utilize a system. The suite can be integrated into the energy measurement track introduced earlier (see Section 3.7.4), to correlate benchmark results with energy readings.

Figure 4.6 depicts the hierarchical architecture of the benchmark. The coordinator is reading the XML benchmark definition file and sends out workload definitions to its clients, which are remotely connected to the

coordinator. Each level in the benchmark may consist of several sub-level instances, each modifying a variety of parameters to run the same or slightly modified workloads repeatedly. Hence, the top node *benchmark* usually contains a number of *series*, each consisting of several scenarios with different parameters, e. g., number of parallel DB clients. Likewise, *scenario* and *scene* are sub-level instances, able to modify other factors, influencing the workload. A scene contains several *settings*, each evaluated in parallel on a dedicated remote client. Hence, by controlling the composition of settings and scences, workloads variations can easily be created.

As Figure 4.6 indicated, *settings* are evaluated at the client side, distributing *workloads* among worker threads. The worker threads finally execute *workunits*, consisting of a series of *actions*, which can be Linux commands to be issued, queries to be sent to the cluster, or scripts to be executed. After executing all workunits, workloads, and settings, the clients report back to the coordinator, where benchmark results are collected.

With this implementation, it is possible to submit repeatable, complex workloads and automatically process results. Since the benchmark definition file is highly customizable, the effect of workload changes can easily be tested against the SUT.

5 The WattDB Framework

In this chapter, we present WattDB, our research prototype of an energy-proportional DBMS. First, we give an overview of the leading design principles, the cluster hardware and the software implementation. Then, we explain some tailor-made optimization techniques, allowing efficient query processing in a cluster of distributed nodes.

5.1 Design Principles

At the beginning of our project, no DBMS existed that met the desired energy-efficiency requirements. Although several matured, open-source DBMS projects were available that could have been extended for energy efficiency, all of them suffered from major drawbacks that would have made it very complicated to rely on them.

Since it would have required an excessive amount of work to implement mechanisms for energy efficiency, we have opted for developing a new system from scratch, tailor-made for our needs.

To gain low-level access to the underlying hardware and reducing runtime overhead, we have chosen *C/C++* as the main programming language for our DBMS. Since WattDB is designed to running solely under Linux, platform independence is not an issue. We are using GCC 4.x for compiling.

5.2 Cluster Hardware

The experimental cluster, on which we ran WattDB, consists of 10 identical nodes. Each node is made of a 1U Supermicro SuperChassis 813MTQ-350CB[1] with a 350 watts, 80+gold-certified power supply

[1] `http://www.supermicro.com/products/chassis/1U/813/SC813MTQ-350C.cfm`

and four 3.5" drive bays. The chassis contain a Supermicro X7SPE-HF[2] main board with Intel Atom D510 CPU and 2GB of DDR2 SO-DIMM RAM.

To connect all nodes, the cluster contains a Gigabit switch (3Com Baseline Switch 2928[3]), with a forwarding rate of 41.7 Mpps, i. e., the switch is capable of processing 41.7 million packets per second. Assuming ethernet packets with 1.500 bytes in size, the switch is able to transfer 56 Gbit/s between distinct nodes. Hence, it should be powerful enough to allow all nodes to pair up and exchange data at wire speed, i. e., 5 pairs of nodes, each sending and receiving 2 Gbit/s of data at full duplex.

Each node in the cluster can be equipped with up to four storage disks. Depending on the experiment, several choices are available to be installed:

Qty	Model	Size
40 x	Western Digital WD2500BEKT	250 GB
10 x	Samsung SSD 830	128 GB
14 x	Samsung SSD 840 PRO	256 GB

5.2.1 Power consumption

The components of the cluster were selected, having their performance and power consumption in mind. To reduce overall idle power consumption, components designed for mobile (battery-operated) use seem favorable. Therefore, we have selected an Intel Atom CPU with low power consumption. At the bare minimum, with no disk drives attached, a single node consumes about 22 watt when idle. With four hard disk drives attached and at full CPU utilization, each node consumes ~41 watt. Table 5.1 gives an overview over the components' power consumption.

[2] http://www.supermicro.com/products/motherboard/ATOM/ICH9/X7SPE.cfm?typ=H&IPMI=Y

[3] http://www.andovercg.com/datasheets/3Com-3CRBSG5293-2900-switches.pdf

Device	Power Consumption
Switch	
	19 - 22 watts
Node	
off	0.5 watts
suspended	2.5 watts
idle	22 watts
100% CPU	26 watts
Storage Disk	
idle	1 - 2 watts
active	2 - 4 watts

Table 5.1: Power consumption per component

5.2.2 GPU

The nodes in the cluster do not have a dedicated GPU attached for two reasons: First, the on-board GPU—integrated into the CPU— is more than sufficient to display the text console occasionally. Second, the servers are operated mostly *headless*, i. e., without a display connected.

Vítor Uwe Reus has examined the use of *General Purpose Graphics Processing Units* (GPGPU) in WattDB in his bachelor thesis [Reu12]. He developed a framework, depicted in Figure 5.1 to execute database query operators on the GPU, and compared performance and energy consumption to traditional execution on the CPU. To integrate GPU operators into the execution pipeline, copies of the records in main memory need to be transferred to the GPU memory for processing. Likewise, results need to be copied back to main memory to let CPU operators consume them. Reus has implemented automatic handling of data transfers to transparently switch from CPU to GPU and back, denoted as *Copy Operator* in Figure 5.1.

In his experimental evaluation, using an NVIDIA GeForce GT430 with 1GB of DDR3 RAM, he compared sorting operators running on CPU and GPU. Reus observed lower query runtimes when sorting on the GPU, especially for large quantities of records, where the query plan using CPU-sort was 3.3 times slower than using GPU-sort.

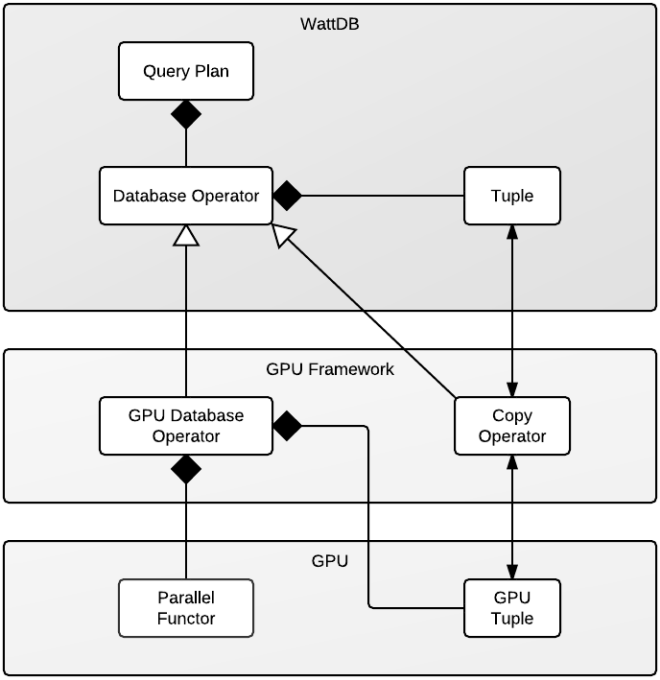


Figure 5.1: GPU framework in WattDB, from [Reu12]

Yet, the addition performance comes at the cost of increased power consumption. Installing a GPU on the node increases its idle power consumption by 11 watts. When utilized, the GPU consumes about 26 watts, practically doubling the node's peak power consumption when running at full CPU load.

While at peak utilization, GPUs exhibit better energy efficiency than the installed CPUs, copying data back and forth is time consuming, and realistic DB workload will leave the GPU underutilized for most of the time. Therefore, their high idle power consumption will drag down the cluster's overall efficiency. As a consequence, we did not include GPUs in all future experiments on WattDB.

5.2.3 Amdahl-balance

The hardware used to build our cluster are commodity components with moderate performance. Especially, when compared to high-end database machines, as presented in TPC-* results, our nodes are wimpy.

An important aspect is the *Amdahl-balance* (see Section 2.1.3) of the nodes and the cluster. Hence, IOPS, processing power, and network bandwidth need to be tailored to avoid introducing a bottleneck, while other components are oversized. While Amdahl originally defined a specific I/O to CPU frequency ratio, we believe that the task defines the exact requirements for both. For example, in the field of databases, a huge difference exists between OLTP and OLAP workloads. While the former require random I/Os on disk to update records, the latter relies more on sequential reads. Thus, the exact ratio between (random and sequential) IOPS to CPU frequency vary.

Therefore, we argue that our cluster of lightweight nodes, equipped with max. 40 storage disks and connected by a moderately fast network, is Amdahl-balanced.

On one hand, this limits the peak performance we may achieve with our cluster, but it likewise limits peak power consumption. Therefore, overall acquisition costs are lower, electrical measurements are easier to implement, and a suitable high-end machine with comparable performance to the 10-node cluster is affordable. Further, by installing components with limited data bandwidth, GB-ethernet wiring is sufficient for interconnecting the node.

On the other hand, we argue that all techniques introduced in this work can be applied to upscale clusters of larger machines, given an adequate interconnect, e. g., InfiniBand. Given a limited budget, a compromise on cluster hardware and size was necessary.

5.3 Software Design

The nodes in the cluster are all running the same version of the database software—*WattDB*. As explained in Section A.1, our implementation was gradually improved to include more and more features. In the following, a few key points, relevant to all experiments in this thesis, are outlined.

5.3.1 Master node

One of the nodes is manually selected to become the *master node*, coordinating all activities in the cluster. This server is the only non-suspendable node in the cluster, responsible for accepting client connections, optimizing queries and managing the cluster's elasticity. On the master node, all monitoring results from nodes are collected and aggregated (see Section 5.4). The master has a holistic view of the cluster, all metadata, and the state of all nodes. Further, the master is coordinating transaction commit and resolving distributed deadlocks.

5.3.2 Communication

All nodes are assigned to the same IP sub-net, allowing them to communicate directly. TCP is used to send messages reliably without having to implement custom packet order and delivery checks. Standard TCP uses *Nagle's algorithm* [Nag84] to reduce the number of small packets by delaying the send operation until either enough data for a full IP packet accumulated, a timer expired (around 200 ms), or the receiving node acknowledged the reception of the previous packet [PD07]. While this approach is well-suited for most applications, it led to significant delays in WattDB. Therefore, we disabled Nagle's algorithm and utilized *TCP_CORK* to suspend and resume putting packets on the wire, as desired.

All nodes in the cluster connect to the master after start-up to receive metadata, queries, and control commands, e. g., *shutdown* requests. The connection to the master is only closed when the node goes down and is immediately re-established after restart. Through this connection, the nodes receive a list of all nodes in the cluster, allowing them to contact the other nodes directly. The master also informs the nodes about metadata changes, i. e., database configuration, storage assignments, etc. Further, the nodes send information about their configuration to the master, and performance data is periodically reported.

To communicate among one another, all nodes may maintain connections to arbitrary nodes. With these secondary connections, queries are handled and records (or storage pages) are exchanged. Typically, these connections are established on demand, whenever one node desires to communicate with another, and never disconnected.

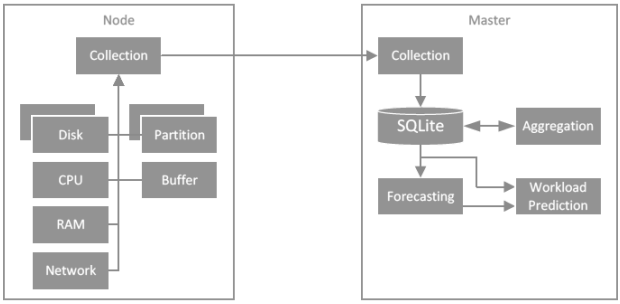


Figure 5.2: WattDB's monitoring framework

Likewise, all nodes may accept connections from database clients to deliver query results, thus bypassing the master node as the final hop for all records. Yet, queries have to be submitted to the master node, which then instructs the client to pick up results from one of the other nodes. These connections are cut when the client disconnects from the database.

5.4 Performance Monitoring

The centralized master node of the cluster needs up-to-date information about the current utilization of all nodes' relevant components. For example, detailed statistics about disk utilization and the share of I/O for each partition are necessary to efficiently distribute data among the nodes. Performance data are also required to detect and react to performance bottlenecks. A monitoring solution needs to run on all (active) nodes in the cluster and non-intrusively measure relevant points frequently.

5.4.1 Physical probes

Physical probes, i. e., *hardware-related performance probes*, reflect utilization of hardware components of the cluster. This type of probes periodically report the components' load to the master, to determine, whether or not the devices' utilization is balanced. For each component,

separate probes and readings are sent to the master. Each component is identified by a unique ID, composed of the node ID the component is connected to and the name of the component. Since the nodes are running Ubuntu Linux, common names for hardware devices are *eth0* for ethernet cards and *sda, sdb, sdc, ...* for disk drives. Hence, a typical identifier for a probe is *node3/sdb*, referencing the second disk of the third node in the cluster. Further relevant probes reflect network and disk load—both storage and IOPS—, and CPU and main memory utilization.

5.4.2 Logical probes

To identify the source of the hardware's utilization, *software-related performance probes* are necessary. Logical probes, also known as *database tracing probes*, monitor utilization of database objects, e. g., tables, partitions, index structures, and buffer statistics.

Similar to physical probes, each logical meter is referred to by a unique identifier. Since placement of logical entities may change over time, identifiers do not carry a node prefix. Instead, they form a hierarchy starting with the table name, followed by the partition identifier, e. g., *CUSTOMERS/p4/ix1* refers to the first index of the fourth partition of the *CUSTOMERS* table.

Further important logical probes measure buffer hit rate, reference count and I/O-rate of partitions, CPU cycles of query operators, and network usage of queries.

By correlating data from both kinds of probes at the master node, WattDB can detect, whether hardware components are running outside their operating window and further, what database components are causing this.

All probes periodically reporting recent figures to the master node are called *counters*. To facilitate quick response times, *events* are sent aperiodically, when certain situations are detected to immediately inform the master node. For example, out-of-memory exceptions and transaction deadlocks need to be resolved promptly.

Readings are stored in an embedded database on the master node for trend analysis and prediction (see next section). Since storing each single measurement—which are performed on a 10-second basis—for extended periods of time would require an immense amount of storage

time	-60	-50	-40	-30	-20	-10	-0
value	50	60	50	40	40	80	60
min	50	60	50	40	40	80	60
max	50	60	50	40	40	80	60

time	-480	-420	-360	-300	-240	-180	-120	-0
value	15	25	20	25	30	30	20	55
min	10	20	20	10	20	10	20	40
max	30	30	20	40	50	40	20	80

Table 5.2: Sample round-robin database

space, data is aggregated as it ages. Without aggregation, measurements from a single node produce around 2 GB of data per year [Dus12]. Predictions and extrapolations add up even more storage space. Besides storage requirements, processing times on large amounts of data slow down workload estimation. Hence, consolidating old data is necessary to reduce the overall amount of information.

Therefore, WattDB uses a *round-robin database* (RRD) [Oet98] to store monitoring data. In an RRD, measurements are stored with a timestamp. While newer data is kept at higher resolution, older measurements are consolidated and stored in more extended intervals. Since aggregation of monitoring results blurs individual data spikes, the minimum and maximum values of all data points inside a consolidated interval are preserved as well. Table 5.2 prints an example RRD with 6 recent measurement values (from time -60 to time 0), where min and max are identical to the value itself, and 6 consolidated values covering larger time spans (-480 to -120), where min and max differ from the (average) value. The latest 7 monitoring results will be aggregated shortly to form the lower-right consolidated value—enclosed in dashed lines—in the RRD file.

5.4.3 Initial implementation

In 2012, Dusso introduced the first version of a monitoring framework for WattDB, sketched in Figure 5.3. In his thesis, he implemented the initial database on the master node to store all measurements. His im-

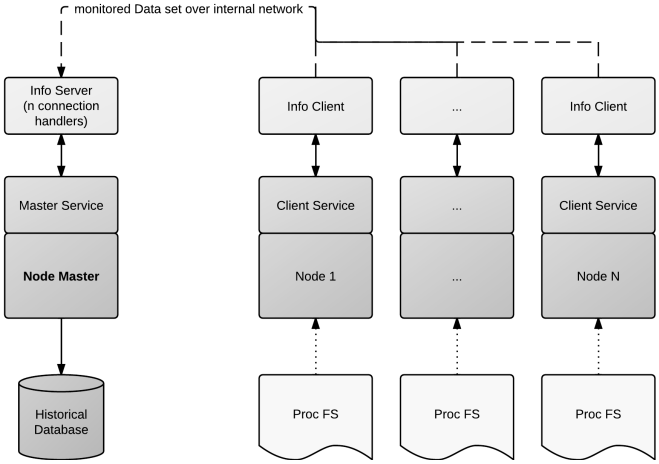


Figure 5.3: Architecture of WattDB's monitoring, from [Dus12]

plementation, presented in [Dus12], is capable of monitoring hardware probes in the cluster, i. e., CPU, main memory, disk, and network utilization by parsing the *proc filesystem* [Ngu04] on the nodes periodically. When enabled, monitoring consumes less than 5% of the cluster's performance (at moderate workloads). Especially under high utilization, effects are negligible.

5.5 Forecasting

Monitoring database workloads gives a detailed picture of current utilization. By archiving the readings, we are able to tell the historic load of the cluster. These data can be used to reactively adjust the cluster to the current workload. Increasing the size of the cluster after detecting an increase in utilization puts additional stress on the nodes. Hence, it is counterproductive to scale-out, when nodes are already overloaded.

Therefore, to be able to proactively configure the cluster, i. e., to foresee workload changes and to prepare for them in time, a dedicated component is needed to forecast upcoming loads by extrapolating from historic data.

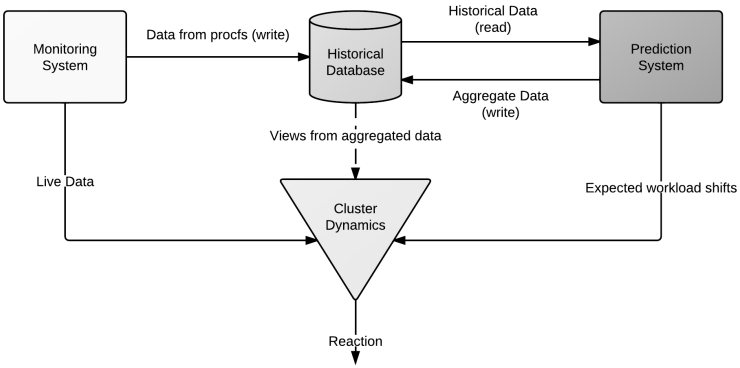


Figure 5.4: WattDB's monitoring & forecasting, from [Dus12]

In his thesis, Kramer presented a forecasting framework integrated in WattDB [KHH12]. His implementation build upon Dusso's implementation of a performance database. Kramer's prediction system reads historical monitoring data and extrapolates expected future patterns, as depicted in Figure 5.4.

Database usage (in productive systems) exhibits certain usage patterns, based on human interactions. Therefore, these patterns are repetitive on a daily, weekly, monthly, or yearly cycle. Furthermore, floating holidays and vacation times typically influence the formation of patterns.

To address these influential factors, Kramer's prediction component performs linear regression on historic data of various granularities, e. g., daily, monthly, and yearly. Each linear regression run generates a distinct usage prediction profile. All results are then combined to form a *weighted average usage profile*. By correlating final results with actual utilization, the prediction can adjust the weights in the future, thus, it is self-adapting.

The prediction component of WattDB increases the size of the monitoring database, depending on the time frame to predict. Typically, for short-term preparation of the cluster, forecasting over the upcoming 60 minutes is sufficient, thus requiring very little additional storage space for predictions.

Prediction, i. e., running linear regression over a plethora of monitoring data, is time-consuming. Since forecasts over long time periods are not required to prepare the database for the near future, we have limited the amount of regression profiles, and excluded monthly, yearly, and all variable holiday patterns from WattDB. Thus, we were able to reduce the runtime of the prediction algorithm, running every minute, to a few seconds on the master node.

5.6 Query Optimizations

Distributed databases with records scattered among multiple machines require distributed query execution to keep processing close to the data. Besides the need for distribution, offloading database operators to a larger number of nodes and, thus, parallelizing query evaluation will also potentially improve throughput and processing time. Although query optimization is not the main focus of this thesis, improving energy efficiency of a badly performing system is futile. Therefore, we present several techniques for improving query execution using multiple nodes in this chapter.

5.6.1 Operator optimizations

In Online Analytical Processing (OLAP) and Map/Reduce, large fractions of data are typically scanned to generate aggregated key indicators, grouped by some attributes. Figure 5.5 shows an exemplary operator tree of an OLAP query. First, records are read from disk and selection and projection operators (σ & π) filter records by predicates. Next, records from different tables are joined using another predicate and results are sorted ($<$), grouped by column (γ), and aggregated (α).

In this example, records read from disk can pass through the first operators (selection and projection) one-by-one, becoming immediately available for the next operator in line. Since selection and projection are able to process records without needing outside knowledge, they can pass on each single record as soon as they finish processing it. Hence, these are called *streaming operators* [Bäc13].

Operators like sorting, on the other hand, need to read all records from the underlying operator before they can emit the first, sorted

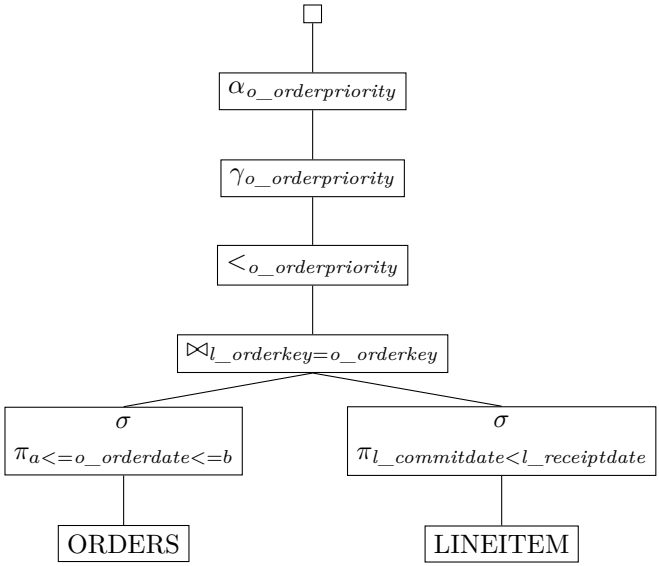


Figure 5.5: Example operator tree for TPC-H query 4

record. Therefore, these are called *blocking operators* or *pipeline breakers* [Neu11].

To speed up query evaluation, distinction between these two types of operators is crucial. Streaming operators can be *stacked* to form a pipeline, where each record is passed from the bottom through all operators until the top—if not discarded on the way. Their memory requirement is constant. However, blocking operators need to cache large amounts of records, requiring temporary (main) memory to store intermediate results, scaling linearly with the number of records to process.

As Bächle points out in his thesis [Bäc13], the placement of blocking operators heavily influences overall performance. Therefore, WattDB tries to put blocking operators on top of a sequence of streaming operators to reduce temporary memory requirements. Further, in distributed query plans, blocking operators are usually the last operators on a node, results are picked up by the next sequence of blocking/streaming operators on the next node to distribute temporary memory demand evenly across the cluster.

```
void open();
bool next(record_t& record);
void close();
```

Figure 5.6: Iterator model

```
void open();
bool next(record_t[]& record, int& num_records);
void close();
```

Figure 5.7: Vectorization

Volcano-style operators

In [Gra94], Graefe introduced a query operator model based on an iterator model, depicted in Figure 5.6. Every operator provides the same interface to callers, exposing the methods *open()*, *next()*, and *close()*. The method *next()* returns a single record on each call, allowing the caller to consume the result and move on to the next record. This approach provides great flexibility in implementing operators and encapsulates the details of each operator from the outside. Yet, the standard iterator model causes a lot of call overhead as, for every single record read from the underlying operator, a new call to *next()* is necessary.

Vectorization

To mitigate the call overhead in the standard iterator model, Boncz, Zukowski, and Nes implemented an improved calling convention, returning a set of records on each call to *next()* instead of a single one [BZN05], depicted in Figure 5.7.

Although this approach requires more intermediate memory to store records and makes query evaluation more complex, it significantly reduces the number of calls while still providing a uniform interface among all query operators.

To control static memory allocation, we extend the method signature by passing an integer value, denoting the maximum number of results to return when calling *next()*. This integer denotes the maximum number

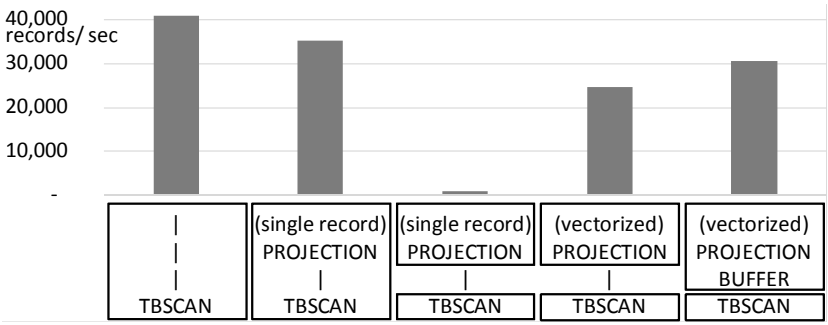


Figure 5.8: Comparison of vectorized and non-vectorized query operator performance

of records returned, but the callee is free to pass less than the maximum for any reason, e. g., if there are no more records to return or if processing ends on a page boundary, making it useful to return earlier to release resources. By selecting a higher number to the callee, more memory will be consumed for intermediate records returned, but call overhead will be reduced, since less calls are needed. Likewise, a small maximum of return values will save memory but increase the number of calls.

Typically, big chunks of records—up but not limited to 10,000—are passed on each call, effectively reducing the number of calls needed by orders of magnitudes.

Offloading

Vectorized query operators are especially useful when distributing query plans on remote nodes, since high network latencies (compared to local call cost), make reducing the number of calls even more important.

In Figure 5.8, a comparison of non-vectorized iterator-based and vectorized iterator-based query execution is plotted. The first bar (from left to right) depicts throughput (records per second) of a non-vectorized iterator-based query operator emitting records.[4] The next bar shows throughput of a vectorized operator, emitting the same set of records.

[4]Records were artificially generated by the operator to mask disk access times or other latencies.

Both operators were running on a single node, hence, imposed by local function call overhead.

The next two operators, again, non-vectorized and vectorized, were running on a remote node, and each call involved network communication (RMI) to fetch records. Here, call overhead is much higher, since typical network round-trip times are orders of magnitude bigger compared to local calls, and throughput is therefore lower. Yet, the vectorized operator performs significantly better than the non-vectorized version. The last bar introduces buffering to the query pipeline, described in the next section.

In comparison, the advantages of vectorization are obvious, especially in distributed environments.

Buffering

Traditionally, operators block while fetching results from their underlying operator and wait for new records to arrive, then process the results (and emit records on their own) before requesting the next batch of records. Thus, calling is synchronous and blocking.

For local query processing, with only a single CPU to run operators, blocking does not hurt overall performance, since either the parent or the child operator are running and occupying the processor. For multi-core architectures, parallel intra-query processing of operators may increase response times. Yet, a typical workload, consisting of a few queries running in parallel, occupies all processor cores using inter-query parallelization and, thus, supersedes the need for intra-query parallelism to keep all cores utilized.

When distributing query plans on remote nodes, another aspect comes into play, when considering asynchronous, non-blocking processing of intra-query operators. Using synchronous calls, each step requires network communication and, thus, impacts processing far worse compared to local call overhead. Even vectorization cannot hide latencies entirely, as the results in Figure 5.7 reveal.

In the left four settings, all calls were blocking and, thus, waiting for new records to become available before processing in the *projection* operator continued. When remotely distributing operators, throughput declines because of network latencies and bandwidth limits.

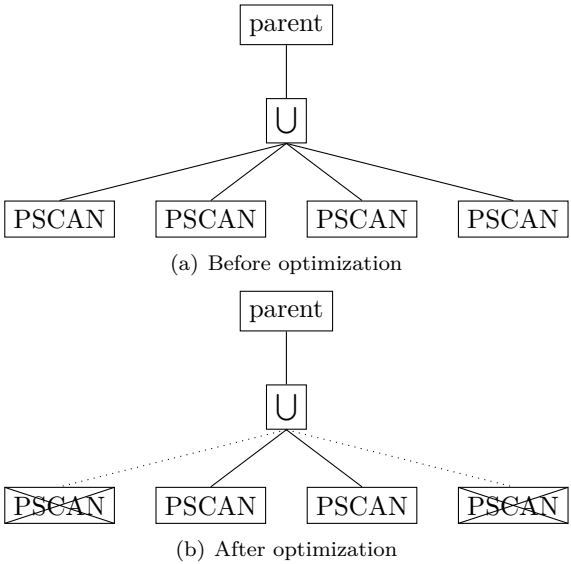
To overcome latencies, buffering can be used, asynchronously prefetching records from an underlying operator while the parent is still processing another batch. In the rightmost bar, a *buffering* operator is placed between the original ones, running on the same node, enabling parallel processing on both nodes without blocking. Compared to the vectorized, non-buffered query plan, throughput is noticeably higher.

5.6.2 Plan optimizations

Distributing data—and query plans—enables parallelization of concurrent operators in the plans on different machines. In theory, super-linear speedup is possible, by running operators independently on separate nodes without synchronization and by exploiting cache effects. For example, reading all records from a table of size T, where $T > BufferSize$ in main memory, requires frequent paging of relevant DB pages. By partitioning data into p partitions of size P, where $P * p = T$, and distributing the partitions among p nodes, the whole data may fit into the respective buffers on the nodes and, thus, eliminate the need for paging, which speeds up processing.

In practice, even linear or sub-linear speedup is desirable when distributing queries, at least for performance-centric approaches. Yet, from an energy-centric point of view, distributing queries on n nodes must yield a runtime reduction by the same factor n to be energetically equal. Sub-linear speedup with the cost of linearly increased power consumption results in sub-linear energy efficiency: Let R_i denote the runtime of a query when running on i nodes. We assume identical power consumption for all nodes utilized for query processing, where P_i denotes the overall power consumption of i nodes. Then, energy consumption (EC) of a query running on i nodes is expressed as $EC_i = R_i * P_i$. Since $P_i = P_1 * i$, the power consumption scales linearly with the number of nodes. Now if runtime scales sub-linearly, hence, $R_i > \frac{R_1}{i}$, in result, $EC_i > EC_1$, i. e., energy consumption for a query running on a cluster of $i > 1$ nodes is higher than the same query running on a single node.

Besides speedup by parallelizing operators, knowledge of the underlying partitioning scheme can be exploited to optimize query plans. In the following, a few of the techniques used in WattDB are explained.



(a) Before optimization

(b) After optimization

Figure 5.9: Partition pruning

Partition pruning

Horizontally partitioning large tables by attribute ranges splits the data into smaller parts. Each record stored in one of the partitions satisfies the partitioning attribute of said partition [Ora08]. Hence, when querying records by partitioning attributes, large fractions of the data can be ruled out immediately, because the range of the partitioning attribute does not match the query's predicate.

To enable partition pruning at query optimization, the query must contain an equals- or range-predicate on the partitioning attribute of the table. The optimizer will then replace all data access operators with a subset of operators, probing only required partitions, that may contain records satisfying the attribute. In Figure 5.9, an abstract example query plan before and after partition pruning is plotted. On the latter, partitions not containing eligible records are automatically removed from the query plan.

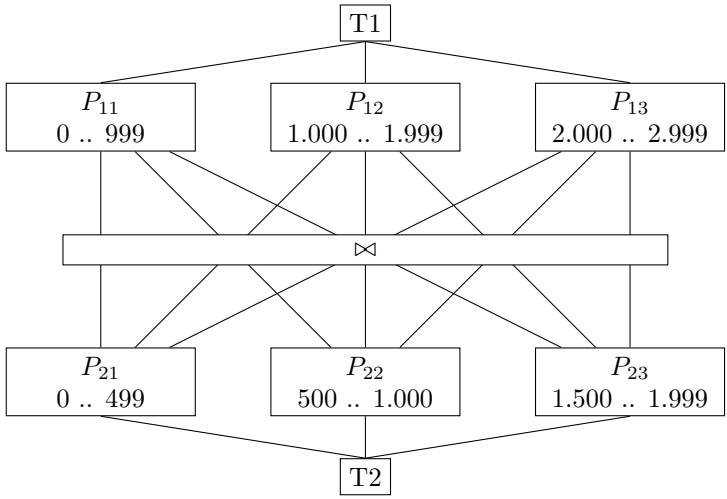
Partition-wise join

Similar to partition pruning, joining two tables on their partitioning key calls for reducing the overal amount of intermediate results, by joining matching partitions only. For example, this technique, called *partition-wise join*, is implemented in Oracle's DBMS [Dij10]. Figure 5.10 depicts a exemplary join plan with and without using partition-wise joins. The tables in this example are both (non-overlapping) partitioned by the join key, hence, matching partitions can be identified easily. The naïve, partition-unaware implementation joins all partitions of both relations with each other. Assuming n partitions for the first relation and m partitions for the second, this results in a total of $n * m$ joins.

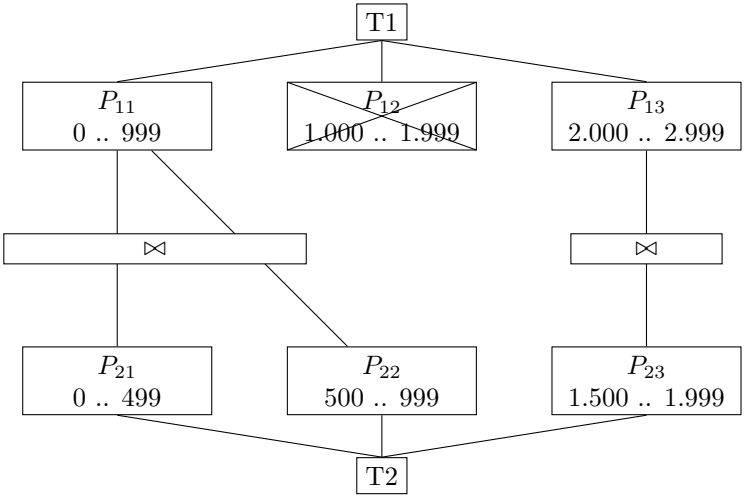
An optimized approach, aware of the key ranges in each partition, can eliminate joins of two partitions with non-overlapping key ranges. Hence, in the worst case—when the key ranges of all partitions in the two relations overlap—each partition can have two join partners at maximum. A single partition is left over with only one join partner, resulting in an upper bound of $2 * max(m, n) - 1$ joins.

Parallel merge-Sort

Analytical queries often require sorting of results, either explicitly by requesting a specific sort order in the query, or implicitly by utilizing sorting to simplify subsequent evaluation, e. g., grouping records by sort attribute. A naïve approach, as depicted in Figure 5.11(a) first gathers records from all partitions and then sorts the results. Since sorting, at best, exhibits a complexity of $O(n * log(n))$—proof can be found in [OW11, p. 154]—, with n denoting the total number of records, sorting huge arrays of records is inefficient. Figure 5.11(b) shows an optimized version, were records from each partition are pre-sorted and then merged to produce the final result. By partitioning n records into p partitions, each containing $\frac{n}{p}$ records, complexity for each part is $O(\frac{n}{p} * log(\frac{n}{p}))$. Merging the records from all partitions exhibits a complexity of $O(n)$ again. In total, the optimized version with $O(p * \frac{n}{p} * log(\frac{n}{p}) + n) = O(n * log(\frac{n}{p}) + n)$ is comparably complex, but sorting individual runs can be done in parallel. Hence, assuming all p partitions are stored on distinct nodes, pre-sorting can be speeded up by p, reducing runtime complexity to $O(p * \frac{n}{p} * \frac{log(\frac{n}{p})}{p} + n) = O(\frac{n}{p} * log(\frac{n}{p}) + n)$.

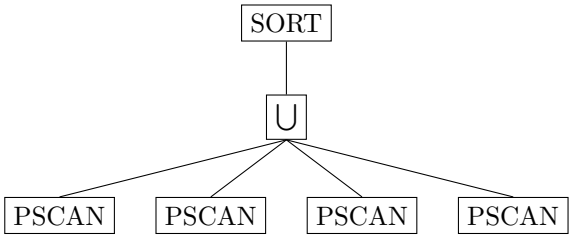


(a) Before optimization

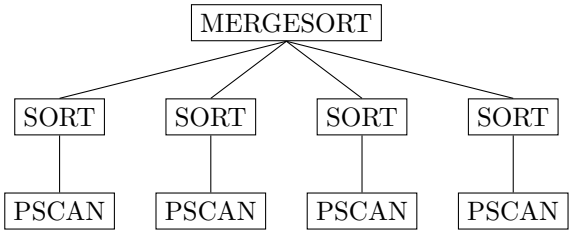


(b) After optimization

Figure 5.10: Partition-wise join



(a) Naïve sorting



(b) Optimized parallel merge-sort

Figure 5.11: Iterator model

Partitioned aggregation

Similar to parallelizing merge-sort, aggregation and grouping operators can also benefit from data partitioning. During query evaluation, records are usually first grouped, before each group is aggregated, independently of other groups. Aggregation drastically reduces the number of records returned to one per group, regardless of the number of input records. In a distributed environment, the standard approach would require shipping all records to a central instance, where they are grouped and aggregated. The overhead of this approach in terms of network communication and bandwidth usage is obvious.

Therefore, an alternative is to first group and aggregate local records and only ship results for final aggregation, similar to *Combine* operators in Map/Reduce [Gat+09].

After evaluating remote execution of operators, we conclude that, for linear (non-parallel) operators, execution on a single machine yields the fastest results. In this configuration, network communication with

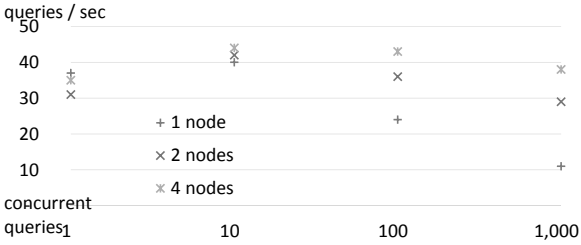


Figure 5.12: Offloading operators: Performance comparison

high delays and low throughput are not present, as records are copied in-memory only. While the best performing configuration for a single query is non-distributed, typical DBMS workloads stress the system with multiple parallel queries. Here, other effects come into play and concentrating query evaluation on a single node may not provide best results.

Operator offloading

A single, lightweight machine is quickly overcharged when a lot of parallel queries arrive simultaneously, as depicted in Figure 5.12. On the X-axis, the number of parallel queries is plotted, increasing from left to right. The Y-axis denotes query throughput per second. All queries consist of an access operator, fetching some records by predicate and a subsequent aggregation. While the access operator needs to run where the data is stored, the aggregation can be offloaded to another node to relieve some workload from the first node. The red indicators show performance for a single node, where a single query plan is evaluated without distribution. Throughput drops quickly as the workload intensifies, since a single node is overwhelmed with the number of queries.

As tested earlier, distributing queries in a cluster reduces record throughput compared to local evaluation. Yet, these experiments were conducted using isolated queries with no contention.

When offloading the higher operators (aggregation, sorting) from the aforementioned queries to remote machines, the local node is relieved of some parts of the workload. In Figure 5.12, the blue and green markers indicate performance for the same query workload, running on a cluster

of 2 and 4 nodes. As the results show, adding more nodes to help in query execution, query throughput can be kept higher, compared to a single node.

In conclusion, as long as a single node is not overloaded, local query execution will yield the highest performance and, thus, distribution is not necessary. As soon as a single node gets overloaded (in terms of processing power or main-memory throughput), distributing queries releases stress from the first node and leads to higher overall performance.

6 Shared Disk vs. Shared Nothing

In the following, we are giving a short overview about the traditional definitions of shared disk and shared nothing. Then, we are going to comment on recent developments in DBMS design and how the meaning of these terms changed. Lastly, we define the terms in the context of our research.

6.1 Traditional Definitions

In [Rah94], Rahm classified distributed DBMS by the criteria *external storage allocation*, *spatial location*, and *machine coupling*, see Figure 6.1.

External storage allocation: Rahm distinguishes between shared external memory, where all machines may access the entire database, and partitioned access, where each disk is assigned to exactly one machine. Using shared persistent storage, access control is needed to synchronize access to database pages among nodes. When using partitioned access, each node is responsible for managing a dedicated part of the data set. Queries spanning multiple partitions require support for distributed transactions. Partitioned storage allocation implies running a shared-nothing DBMS. Database systems utilizing shared external memory are at least sharing disks, hence, the architecture is either shared disk or even shared everything.

Spatial location: Shared storage allocation implies, database machines are running closely to each other, hence, in the same room, and are connected to the same network segment. Shared-nothing architectures, however, may run spatially distributed, in distant locations.

For our research, we are focusing on machines running closely together, in immediate vicinity. Therefore, we are not going to get into details of spatially separated databases.

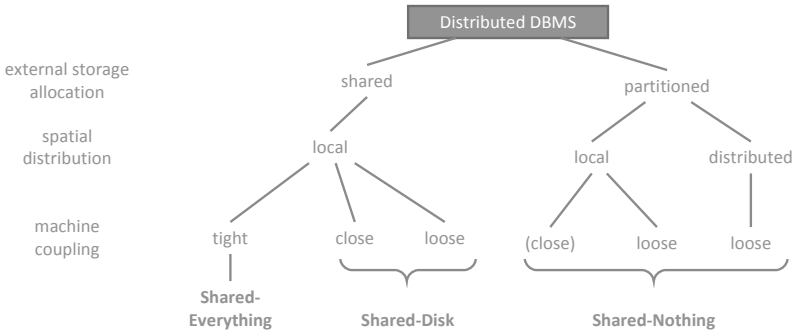


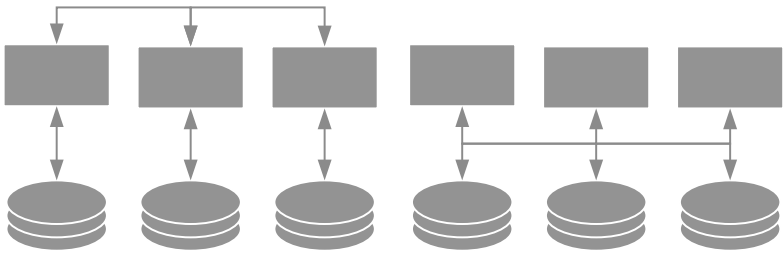
Figure 6.1: Classification of distributed DBMS, according to [Rah94, p. 29]

Machine coupling: Machine coupling describes the way the nodes in the database cluster are coupled and what resources, besides persistent storage, they share. Tightly coupled machines are even sharing processors and main memory among each other. Therefore, only a single instance of operating system and DBMS is running on the nodes; the architecture is called *shared everything*. Traditional shared everything consists of a fixed number of tightly coupled nodes, running a carefully tuned DBMS instance to support the specific cluster configuration.

For the purpose of an energy-efficient DBMS, this architecture is not well-suited, as it does neither offer scale-out support, nor does this architecture allow a fast scale-in to minimize energy consumption at low utilizations.

Loose coupling describes several DBMS instances running on dedicated nodes, without shared memory or processors. Each instance is running on its own, which increases fault tolerance and also simplifies cluster expansion. Since each node comes with its own hardware, the size of the cluster can easily be adjusted by adding and removing nodes. On the downside, communication among nodes requires expensive message passing, which is, compared to shared main memory, slower.

Close coupling tries to mitigate the drawbacks of loose coupling by introducing some shared memory into a loosely coupled system. This memory can be used for global states and synchronization information,



(a) Shared-nothing (b) Shared-disk

Figure 6.2: Shared-disk and shared-nothing architectures compared

thus reducing communication latencies. Yet, with reintroducing shared hardware, the scaling and fault-tolerance issues of the tight coupling come back.

Based on the definitions by Rahm, shared disk corresponds to an architecture, where each node in a DBMS cluster has access to all database pages. Since two nodes may alter the same database page, special care must be taken to ensure cache coherence among all machines.

Contrary, an implementation where disks belong to dedicated nodes, is called shared nothing. In this architecture, nodes may still access remote pages, but the hosting node has ultimate control over access and write-back of the page to disk. Since pages are shipped by explicit messages, the controlling node may decline or delay page requests. Further, a single instance is responsible for integrity control over the owned pages.

Figure 6.2 exhibits both architectures. As the Figure shows, shared-disk approaches feature a communication network between the nodes and the disks, to enable direct access to all DB pages. In shared-nothing architectures, this communication network is missing, and messages are exchanged between nodes; hence, the communication layer resides above (or between) the nodes.

Hence, the distinction between shared disk and shared nothing becomes blurry when considering recent changes in DBMS architecture. Hybrid approaches, where only parts of disks are shared by nodes, are not covered by these definitions.

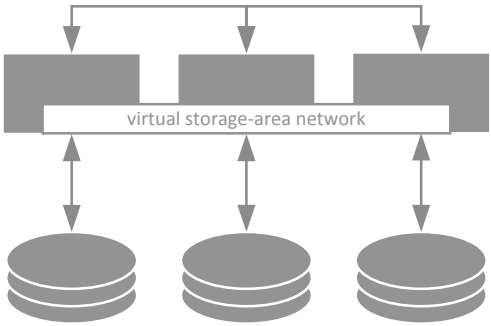


Figure 6.3: WattDB's virtual storage-area network

6.2 Shared-* in WattDB

Shared everything is obviously not well suited for an elastic DBMS, since its tight coupling of resources prevents dynamic scale-out. Therefore, we did not explore this architecture type at all.

Traditional definitions of shared disk and shared nothing do not account for recent developments, therefore, we have to clarify the terms at different layers of the access path.

WattDB is a database cluster of independent nodes, each equipped with storage devices. Nodes may communicate via Ethernet packets, as described in 5.2. Therefore, no communication network between nodes and disks exists, enabling access of all nodes to arbitrary disks. From that perspective, according to the definitions of Rahm [Rah94], this represents a classical shared-nothing architecture.

Although the physical hardware configuration implies a shared-nothing approach, the software implementation of WattDB ultimately defines possible access paths. Even with no storage-area network, it is possible to allow all nodes access to all disks by creating appropriate access mechanisms in software. By implementing a virtual storage network, running distributedly on the nodes, as depicted in Figure 6.3, a shared-disk system can be emulated.

6.2.1 Virtual storage network

To enable shared disk on a shared-nothing cluster, all nodes must be able to access every page on every disk in the system. The address of a physical database page is a composite of three components: First, the node ID, the page is stored on, next, the corresponding disk ID, and, last, a page offset on the disk:

$$PhysicalPageID := NodeID \bullet DiskID \bullet PageOffset$$

With this addressing scheme, each page in the cluster is uniquely identifiable.

Each node in WattDB has direct access to its attached disks. Accessing pages on remote nodes involves network communication. First, a request is sent to the (home) node, including the page identifier. Then, the receiving node loads the page from disk and sends it to the requesting machine. Because this approach simulates a shared-disk solution, the remote node does not cache the page it just read from disk (via its home node). Caching is performed at the requesting node, as sketched in Figure 6.4.

Virtualizing shared disk on a shared-nothing cluster does no longer adhere to the definitions by Rahm [Rah94]. Therefore, we are referring to our solution as a *hybrid storage model*, combining shared disk and shared nothing. By eliminating the previously needed, dedicated storage network to share disks and integrating this function into the nodes, we can reduce the overall energy footprint of a shared-disk DBMS. Further, our approach brings storage closer to the cluster, especially reducing access latencies to disks attached directly to the node. Yet, reading and writing remote pages involves network communication.

6.2.2 Storage segmentation

As with traditional shared-disk systems, the problem of cache coherence among multiple nodes, sharing the same database page, exists in our hybrid storage architecture. In traditional shared-disk implementations, access to pages is not constrained, therefore, every node may access any page. Expensive latching protocols are necessary to prevent chaos during parallel update operations. Further, to keep a page's copies in all

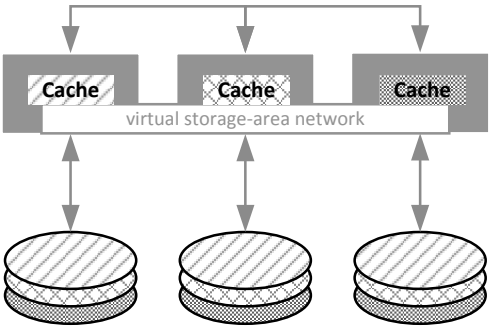


Figure 6.4: WattDB's disk segmentation

caches coherent, a signaling protocol is needed to inform all respective nodes of changes [Rah94].

The advantage of each node having access to the whole database in shared-disk architectures are shadowed by these limitations. For our requirements, unrestricted access is not necessary.

To eliminate the need for latching and cache coherence, we have segmented the virtual storage space into smaller parts, assigned exclusively to nodes, as depicted in Figure 6.4. The differently colored nodes have exclusive control over the data pages stored on the accordingly colored segments on disk. A centralized instance acts as bookkeeper, issuing dedicated segments to nodes upon request. This instance keeps a list of all segments in the cluster and tracks assignment of nodes to segments.

Prior to reading pages from disk, a node first has to request access from the bookkeeper. Access grants do not have to be renewed, hence, the overhead for this approach is minimal. Only when needing additional storage space, nodes need to contact the central instance, asking for additional segments. Likewise, nodes have to inform the bookkeeper of segments no longer needed, to make them available again.

This restriction prevents nodes from accessing arbitrary pages, yet still allows to reassign storage for elasticity.[1]

By virtualizing segment allocation on disks and centralizing access control, we are also able to dynamically move segments among nodes.

[1] Technically, we convert our virtual shared-disk system back to shared nothing.

To migrate a segment from one node (and disk) to another one, the bookkeeper has to inform the node currently accessing the segment about the pending change in order to suppress updates. By submitting a copy request to the receiving node, all pages in the segment are copied to the new location. Afterwards, the node currently accessing the segment is informed of the change, its mapping is updated and pending writes can continue.

This technique allows the cluster to dynamically distribute data among available nodes and to react to workload changes, as we can scale-out and back scale-in without disrupting page access operations.

The bookkeeper can implement various segment assignment strategies. In order to keep latencies low, assignment of segments local to the requesting node is favorable. Yet, to increase IOPS, striping segments among multiple disks on multiple nodes may also seem viable. Ultimately, storage space restrictions have to be respected.

6.2.3 Conclusion

Stonebraker analyzed various aspects of the the three architectures (shared everything—which he called shared memory—, shared disk, and shared nothing) and concluded, that shared nothing is the preferred approach for large-scale, distributed databases [Sto86].

With our hybrid storage architecture, we are able to implement an elastic storage system in a cluster of nodes without additional storage-area network hardware. By abstracting from traditional shared-disk/shared-nothing architectures, advantages from both approaches can be combined in our solution. This is one major building block in implementing a fully elastic, energy-proportional database cluster.

7 Elastic Storage

Dynamically adjusting the size of the underlying DB storage to fit workload demands is the first step on the way to an energy-proportional DBMS. Here, we present our first naïve implementation of an elastic cluster, mainly operating on the storage layer; focus lies on storage elasticity, hence, the number of active nodes providing storage space will be dynamic, while the number of nodes processing queries is disregarded.

7.1 Elasticity in the Storage Layer

We decoupled storage from processing, by introducing an indirection between logical database page addresses and their physical storage, as depicted in Figure 7.1. Hence, the physical storage location of pages may change transparently to upper DB layers.

By separating storage and query processing, as implied by the *5-layer model* [Här05], data storage is decoupled from data ownership,

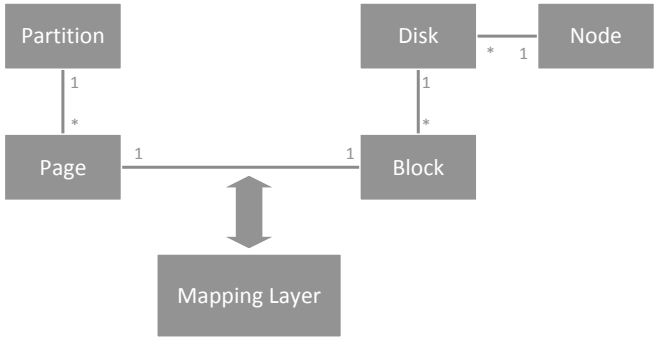


Figure 7.1: Decoupled storage by mapping logical pages to storage blocks

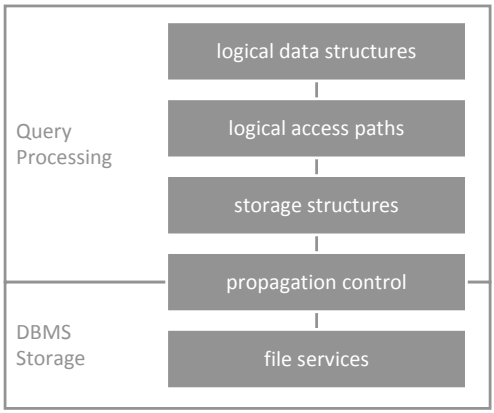


Figure 7.2: 5-layer database architecture

see Figure 7.2. Therefore, both layers can scale independently without impacting each other. First, we have focused on energy efficiency in the storage layer by optimizing page placement w. r. t. storage and I/O requirements [SH13b].

7.2 Design Overview

Database pages are aggregated into segments of size 32MB (4096 pages), which are then mapped to storage blocks. The mapping can be arbitrary, hence, every node in the cluster may access every block on any disk of all other nodes.

In a traditional shared-disk DBMS, all nodes may access any database page at any time, hence, heavy-weight synchronization and locking is required to prevent race conditions. Further, a synchronization protocol must ensure that all nodes have the most recent page version in their buffers. With an increasing number of nodes, this approach requires very fast interconnects and excessively suffers from lock contention [Rah94].

In our implementation, we refrained from this fully-shared approach and set up an allocation instance on a single node to control assignment of blocks to nodes. Though it is still possible for every node to access

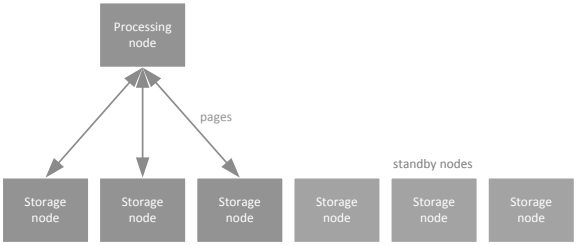


Figure 7.3: Shared-disk design overview

any part of the physical storage, access to blocks is now controlled in order to exclusively assign (parts of) storage to nodes.

By abstracting from a traditional shared-disk solution, we have eliminated the need for locking and synchronization among the nodes, without sacrificing flexibility, as explained in Section 6.1.

The DBMS design is sketched in Figure 7.3: Dedicated machines, called *storage nodes* provide DB storage to the cluster. So-called *processing nodes* perform query processing and fetch required pages remotely from storage nodes, which act as *page servers* [Här+95].

Elasticity is achieved by dynamically redistributing storage blocks to optimize utilization among all storage nodes while minimizing the total number of nodes running. A management component is monitoring disk utilization of all nodes and adjusts the assignment of segments to nodes if I/O thresholds are exceeded.

Listing 1 sketches this process in pseudo-code. First, all active storage devices in the cluster are examined and the current IOPS are compared to the threshold of max. allowed IOPS for this device (line 7 of the listing).[1]

If the current utilization of a device exceeds the threshold for more than three consecutive measurements, it is considered overloaded and the data is distributed to other storage devices. Isolated peaks in utilization are discarded and do not lead to redistribution. Not depicted in this listing is the selection of the distribution targets (line 14): The algorithm tries to move blocks to active, non-overloaded storage devices

[1] The threshold is set to 90% of the peak IOPS for the drive, which was determined beforehand.

Listing 1 Power-management pseudo code

```
1  ForEach(Storage storage in Cluster.Storages) {
2
3     If(!storage.PoweredOn) {
4        continue;
5     }
6
7     If(storage.IOPS > MAX_IOPS_PER_DISK) {
8        // Storage overloaded,
9        // acquire new storage and distribute data
10
11       Storage storageNew = PowerUpAnotherStorage();
12       Storage storageOld = GetStorageWithHighestLoad();
13
14       distributeBlocks(storageOld, storageNew);
15    }
16
17    If(storage.IOPS < MIN_IOPS_PER_DISK) {
18       // Storage underutilized,
19       // consolidate data to other active storages
20
21       consolidateStorage(storage);
22       storage.Suspend();
23    }
24 }
25 // Suspend unused nodes
26 ForEach(Node node in Cluster.Nodes) {
27    If(node.ActiveStorages == 0 &&
28       node.Partitions == 0 &&
29       !node.IsMaster) {
30       node.Suspend();
31    }
32 }
```

attached to the same node first, to minimize network traffic. In case these storage devices are already utilized too much, additional disks on the same node are powered up and used as a target, if possible. Lastly, when all the storage devices identified above are not sufficient to handle the load, other nodes are taken into account as well, and data blocks are shipped over the network to re-distribute the load. In case no other eligible nodes are found, additional storage nodes have to be powered up first.

After analyzing overutilized storage disks and distributing their load, the algorithm now examines underutilized storage devices and tries to consolidate data blocks to other storage devices (line 17). This step performs the opposite work as sketched above and aims to reduce the number of storage disks, while still maintaining sufficiently high IOPS.

Consolidating storage disks (line 21) follows a similar logic as before: First, disks on the same node are selected as target devices, if they are not overloaded already. Second, remote locations are involved and blocks have to be sent via the network. In both cases, *all* blocks are moved to other locations in order to shutdown the originating disk. After redistributing the disk load, the algorithm takes a final step and suspends all nodes, which do currently not serve a purpose (line 26–32).

7.3 Experimental Evaluation

To estimate energy savings and performance impact, we have run a series of benchmarks on our hybrid shared-disk/shared-nothing implementation of WattDB. Complex query capabilities are not yet implemented, therefore, OLTP queries cannot be used to benchmark the database. Instead, the benchmark consists of a set of threads, executing an OLTP simulation.

7.3.1 Simulating OLTP queries

Each thread is representing one database client, running a series of OLTP queries.

Each query consists of a series of page reads, (artificial) processing steps and writes to simulate an OLTP query trace and to generate load at the storage layer; the processing nodes are not utilized much. Hence, this benchmark is heavily I/O-intensive to empirically evaluate especially the storage layer and its energy-efficiency potential.

The benchmark operates on a 128GB database with a primary-key index stored as a B*-tree. The database is preallocated on a single storage node which also contains the index. To circumvent the OS file system buffers and minimize the management overhead, no file system is used; instead, WattDB operates on raw disk devices. The database pages contain multiple records (which currently consist of an ID column and additional columns, filled with *junk* data to increase size). The inner leaves of the index fit into 2GB of main memory, hence, after warming up, the buffer should contain a large fraction of the index. The (simulated) OLTP clients randomly select IDs for reading records. By default, 80% of requests are falling within a 20% range of the data.

For each request, the DBMS traverses the primary-key index to fetch the respective leaf page and locates the requested record inside the page. To emulate data processing, the threads generate CPU load by spin-locking. Finally, with a 1:4 chance, the page gets marked dirty in the buffer and, hence, has to be written back to the storage node at some time.[2] Afterwards, the benchmark thread goes to sleep for a specified time interval, before commencing the next read-process-write cycle. Such breaks are necessary when running an energy benchmark—as opposed to a performance benchmark. The SUT utilization can be tuned by adjusting the number of concurrently running clients, i. e., threads.

7.3.2 Initial test run

We have deployed the WattDB software with an energy management component in the cluster, connected to an energy measurement device. By running the benchmark against a cluster configuration, we expect the software to react to the changing workloads and power up/down nodes as needed. To make results comparable, we have run the identical load profile (as explained by Figure 4.4) three times. For the first cluster configuration, we distributed the DB pages to 2 storage disks (HDDs) on 1 node and disabled the power management algorithm. Therefore, the number of nodes was fixed to the bare minimum of 3 (master, processing node and storage node) and the cluster was fixed to its most power-saving configuration delivering the lowest performance.

As second cluster configuration, we distributed the DB pages to 10 disks on five nodes, and started the same benchmark, again with disabled power management. This time, all nodes (the master, one processing node and five storage nodes) were active and the cluster was able to work with maximum performance and, as a consequence, maximum power consumption.

The results of these two cluster configurations were used as baselines to estimate the performance and power coverage the cluster can achieve. Finally, we set up an unrestricted cluster configuration, with the power management component in full control of the cluster and its current workloads. We expected the cluster to adapt to the current workloads, as the benchmark runs proceeded.

[2] The buffer decides which pages and when to write back.

During each of the runs, we measured the runtime and the energy consumption of the query phase from the start of the first query until the last query finished. Initially, the benchmarks were run on a cluster of seven nodes with 10 magnetic disks attached. Five nodes were acting as storage nodes, each having two disks attached, one was used as processing node and the remaining one was the coordinating master node.

7.3.3 Results

While running the benchmark against the three cluster configurations, power consumption and runtime were reported to file.

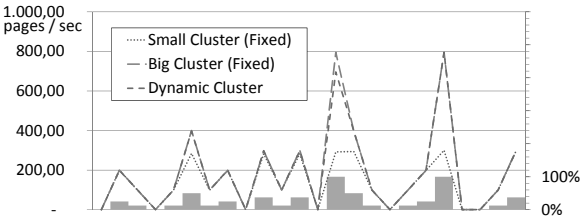
Results from the three initial test runs are graphed in Figure 7.4. In the lower part of each subfigure, the benchmark's workload intensity is depicted.

Performance

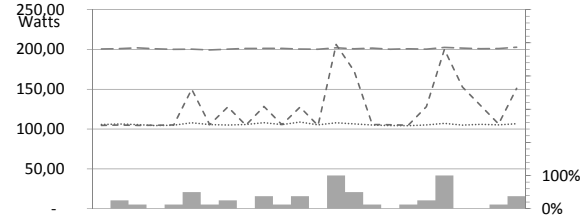
Figure 7.4(a) plots the DBMS' performance while running OLTP simulation traces. The X-axis denotes the time line of the benchmark, while the Y-axis illustrates pages per second. As expected, the big cluster delivered the best performance and was even able to handle the highest utilization. The small cluster's performance broke down, due to it's constrained number of storage disks and the limited maximum IOPS. Finally, the dynamic cluster showed more or less identical performance to the big one, with small limitations in a case where dynamic adaptation caused some blocks to be moved between storage devices, which decreased the maximum performance. (It took only a few minutes to redistribute several Gigabyte of storage via Gigabit-Ethernet. By using compression, we were able to further reduce network traffic.) Compared to the total runtime of at least 30 minutes, redistribution cost was acceptable.

Power consumption

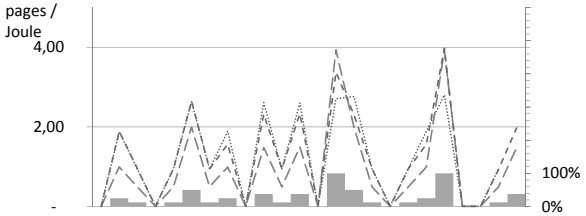
Figure 7.4(b) depicts the system's power consumption during the benchmark runs in watts. Both fixed configurations exhibit a mainly static power consumption, because the number of nodes was fixed. The big



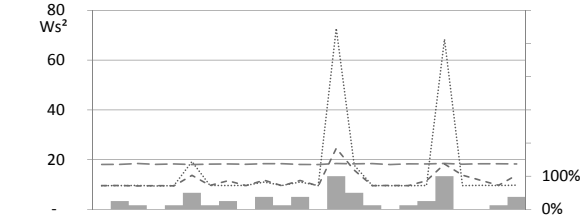
(a) Performance



(b) Power consumption



(c) Energy efficiency



(d) Energy delay product

Figure 7.4: OLTP trace on a dynamic cluster

cluster delivers no measurable difference for the HDD configuration between idle and full utilization.[3] The dynamic configuration oscillates between the lowest and highest power consumption, as the cluster adapts to the workload.

Energy efficiency

If we relate performance to energy consumption (which is power consumption times benchmark duration), we can calculate the energy efficiency for each of the runs, which is shown in Figure 7.4(c). Energy efficiency is expressed in pages per joule, i. e., how many pages can be processed by consuming one joule of energy. Not surprisingly, the small cluster exhibits the best energy efficiency during low utilizations. The big cluster is simply overprovisioned to satisfy the workload and consumes more energy to process the same amount of work, hence, its energy efficiency is worse.

At full utilization, the situation turns in favor of the big cluster. The small cluster is not suited to handle the high utilization and needs almost 3 times as long as the big cluster to process the workload (not depicted here). As a consequence, the energy consumption of the small cluster is much higher and the energy efficiency accordingly lower.

The dynamic cluster powers storage nodes up and down according to the current workload. Therefore, under low utilization, its energy efficiency is identical to the small cluster. With rising load, the dynamic cluster powers up additional storage devices; hence, its energy efficiency gets comparable to that of the big cluster. Again, transition costs to move storage blocks decrease the energy efficiency in the dynamic case.

Energy delay product

Figure 7.4(d) shows the EDP for the three benchmark runs. The small cluster exhibits the lowest EDP under low utilization, but does not perform well under heavy load. Starting at 50% utilization, the EDP of the small cluster outgrows the big cluster's EDP, because the load is too high for the small cluster and the execution time nearly triples. The big cluster shows a stable EDP, regardless of the workload. In most cases, the cluster is underutilized and, thus, more energy is consumed

[3] CPU-bound benchmarks might reveal different power characteristics.

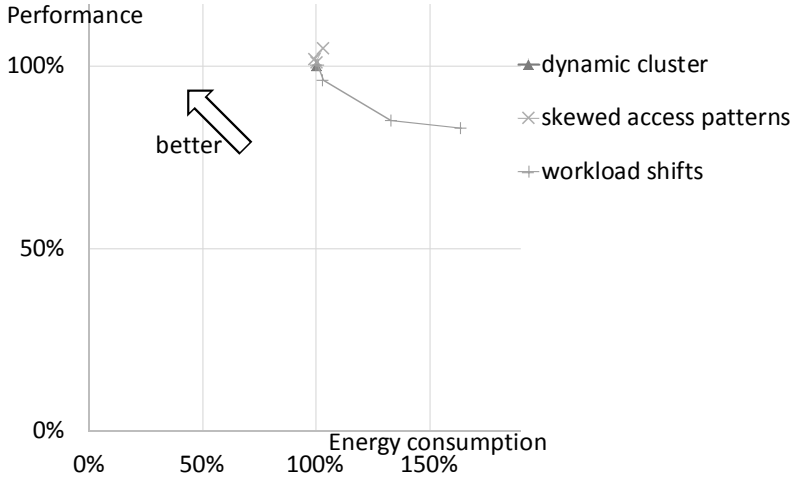


Figure 7.5: Effects of access-pattern variations

and the EDP is higher, compared to the small cluster. Only in peak-load situations, the additional performance of the big cluster pays off. The dynamic cluster shows the best overall EDP, with a similar score to the small cluster when not fully utilized and a slightly higher EDP in the peak-performance benchmarks.

7.4 Skewed Access Patterns

In the initial runs, we have employed an 80/20 access pattern, hence, 80% of all requests go to a set of 20% of the pages. An adversary access pattern, with a higher concentration of requests to even fewer pages might disrupt the cluster's elasticity. Therefore, we have repeated the benchmark on a dynamically adjusting cluster, with skewed access patterns, focusing 80% of requests to 10% (5% respectively) of pages. Condensed results are plotted as orange data points in Figure 7.5. The reference point with 100% energy consumption and 100% performance is the dynamic cluster from the previous experiments. Altering the access pattern to focus on less pages does not have an noteworthy effect on energy and performance.

	Energy Consumption *[joules]*	**Performance** *[pages/s]*
big static cluster	1,470,930	189
small static cluster	777,095	159
dynamic cluster	925,113	187
skewed access patterns		
80/20	925,113	189
80/10	952,866	196
80/5	915,862	191
workload shifts		
hot spot not changing	925,113	187
..changing every 20 min	934,364	187
..changing every 10 min	952,866	180
..changing every 5 min	1,230,400	159
..changing every 1 min	1,517,185	155

Table 7.1: Effects of access-pattern variations

7.5 Workload Shifts

The previous experiments randomly selected pages to read—adhering to the defined 80/20 distribution. The *hot set*, i. e., the 20% of frequently accessed of pages, did not change over time. In the following experiments, we altered the hot set over time, to examine the effect of workload shifts on the cluster. We have varied the speed of change to investigate the cluster's ability to adapt to changing workloads. In Figure 7.5, results are depicted in light-blue.

Contrary to skewed access patterns, workload shifts do influence the cluster's elasticity. As the results indicate, the faster the hot set changes, the more energy is consumed by the cluster to adjust itself to the workload and the less performance is achieved. Yet, even a highly fluctuating workload, where the set of most frequently accessed pages completely changes every minute, the cluster delivers 84% of performance, compared to non-changing workloads, while consuming 164% of energy.

Table 7.1 summarized the results from the last benchmarks on skewed access patterns and workload shifts. The first column lists overall energy consumption in joules for each benchmark run, while the latter column plots average performance in pages per second. Since the workload contained alternating phases of high and low utilization, performance figures only exhibit moderate values.

7.6 Optimizations

Building on the initial page server, we have implemented several optimizations to speed up query processing. Performance of the storage layer is mainly dependent of two components. First, the I/O bandwidth and latency of the connected disks. This parameter can be tuned by modifying models used and the overall number of disks. Second, the connecting network plays a major role in quickly delivering pages to the processing subsystem. Again, bandwidth and latency are crucial factors. While switching from one technology, i. e., Ethernet, to another, faster one, e. g., InfiniBand, is out of the question, configuration of the existing network can be improved.

7.6.1 SSDs

By trading HDDs with SSDs, access latency to storage blocks should drop from about 8 ms on disks to « 1 ms on flash. Yet, overall latencies have to include the software stack of processing and storage nodes, and the interconnecting network. Hence, to assess the total benefit, we have repeated the initial measurements and replaced the HDDs with SSDs. Also, after evaluating the performance of the SSD-based cluster, we have reduced the number of storage nodes to 4, to limit overall performance gains from replacing HDDs to a factor of 10. Figure 7.6 illustrates benchmark results on an SSD-based cluster. The resemblance to previous benchmarks on an HDD-based cluster are obvious.

Performance

Performance using SSDs was much better, compared to magnetic disks. Still, the relative results are comparable, except for the dynamic con-

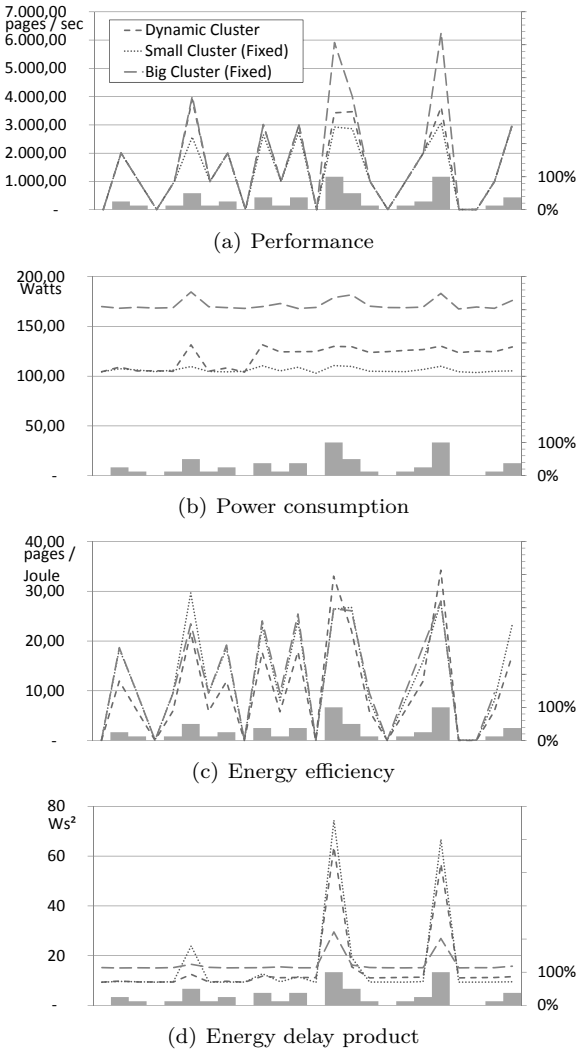


Figure 7.6: OLTP trace on a dynamic SSD-based cluster

figuration, which did not deliver the same peak performance as the preconfigured big cluster. When stressing the SSD cluster with heavy load, the performance of the cluster did not increase as expected. This might indicate optimization potential in the power management component or a bottleneck which was not monitored, e. g., CPU or network.

Power consumption

Using HDDs, the power management decided for our benchmark to use all available storage devices to share the load. For SSD configurations, however, not all storage devices were used, possibly because the storage was not overutilized and some other component of the cluster was the bottleneck. Our power management decided to distribute the load to two storage disks on two separate nodes, instead of two disks on the same node. This is another indicator that the network was the limiting factor in the benchmarks, and not the IOPS of the SSDs.

EDP

Running the benchmark on SSDs, even the big cluster seems to have trouble handling the heavy workloads, hence, the rising EDP. The dynamic cluster is also unable to adjust the configuration to score a low EDP. As previously mentioned, this indicates a bottleneck beyond the reach of the current monitoring.

Overall, replacing HDDs with SSDs seems like a promising option to increase performance while keeping power consumption steady. Yet, the results indicate that demanding workloads seem to run into a bottleneck, when switching to SSDs.

7.6.2 Page compression

The initial implementation, where processing nodes fetch pages from remote nodes, generates high network traffic. To reduce the bandwidth requirements, page compression seems a viable option. Yet, (de)compression requires additional CPU resources to deflate and inflate data and increases main memory needs. In WattDB, we are using the *LZO library*[4], which delivers moderate compression rates and fast

[4] `http://www.oberhumer.com/opensource/lzo/`

compression: none			
storage nodes	processing nodes	pages / sec	CPU utilization on processing node
1	1	8,161	5%
2	1	14,855	7%
4	1	26,939	9%
8	1	27,107	10%
compression: lzo			
storage nodes	processing nodes	pages / sec	CPU utilization on processing node
1	1	7,902	9%
2	1	14,383	15%
4	1	28,974	26%
8	1	39,481	33%

Table 7.2: Effects of compression on performance

decompression. Compared to other compression libraries, e. g., *zlib*[5], LZO delivers sightly worse compression ratios, but does require less CPU and memory [Gil02]. Further, *LZO* offers fastest decompression times. Since database pages are often sparsely filled, a not-so-complex compression algorithm should be sufficient to significantly reduce network overhead.

Table 7.2 lists results from micro-benchmarks, sending pages back and forth between storage and processing nodes. In the first runs, no page compression algorithm was used to create the baseline case. To stress the cluster, we have connected up to 8 storage nodes, each equipped with two SSDs and a single processing node, requesting and receiving pages from all storage nodes. Since randomly reading from SSDs might already impose a performance bottleneck, as revealed in [HS11b]), the processing node is requesting pages sequentially from disks. Yet, the results indicate, a single storage node can only provide ~8,000 pages per second. By adding more storage nodes, throughput can be increased up to 27,000 pages/second, transmitting ~850 MBit/s over the network. This seems to be the throughput limit for a single processing node.

[5] `http://www.zlib.net/`

In the lower half of Table 7.2, the same benchmark is run with LZO compression enabled. Hence, storage nodes compress pages prior to sending, and the processing node is decompressing the page after receiving. Now, with compression enabled, CPU utilization on the processing node rises significantly higher, due to the additional overhead of decompressing each page. Yet, more pages can be sent per second, up to 39,000 in peak, improving total bandwidth by 44%. In theory, 1,250 MBit/s are sent to the processing node, exceeding raw Gigabit-Ethernet speed by 25%.

Yet, to gain this much throughput, the storage subsystem needs to supply sufficient IOPS from disks. With 8 SSDs, each reading sequentially, throughput requirements are met in this benchmark, but are unlikely to appear in realistic workloads. Further, I/O concentration on a single processing node is implausible for a well-configured cluster.

Although these micro-benchmarks exhibit impressive throughput gains by using compression, the additional CPU overhead and the unrealistic requirements to pay off kept us from including this technique in WattDB.

7.7 Summary

All experiments presented in this chapter focus on elasticity in the storage layer, one important building block of a fully elastic DBMS.

In summary, the measurements clearly illustrate that no fixed cluster configuration, neither a small one, nor a big one, is able to process the given workload in the most energy-efficient way. Hence, the results of the dynamic cluster can be considered as a proof of existence that, in specific cases, energy proportionality can be approximated for DBMS processing and that the increased effort pays off in terms of energy saving—without sacrificing too much performance. Replacing traditional hard disks with SSDs does provide better energy efficiency and allows to reduce the number of storage nodes. Yet, fluctuating workloads require elastic cluster adjustment to balance energy consumption and performance.

8 Elastic Query Processing

After evaluating possible energy savings in the storage layer, we have focused on making query processing more energy efficient. Since each single node in our cluster is lightweight, it does not offer many resources for executing sophisticated queries. Distributing the query workload to multiple nodes, potentially scaling the number of machines to the current demand, seems a promising approach.

8.1 Elasticity in the Query Processing Layer

As concluded in the previous sections, elasticity in the storage layer can be achieved by distributing database pages transparently to the upper layers to balance power consumption and performance. Yet, processing in the previous experiments was simulated by OLTP traces, running on a single node, acting as the processing layer of the entire DBMS.

By applying similar design principles as before to the query execution layer, the number of machines involved in query processing may also adapt to the workload, and we may be able to implement elastic query processing in our DBMS.

Using this approach, we do not shift storage pages, but dynamically place queries on an elastic cluster of nodes.

8.1.1 Design overview

In Figure 8.1 our initial design of an elastic query processing layer on a shared-disk storage architecture is sketched. The storage layer, at the bottom, consists of a fixed number of nodes, acting as page servers to the upper query processing layer.

Processing nodes do not hold database pages on local disks, instead, they need to fetch all data from storage nodes. Hence, all processing nodes share all data of the storage nodes.

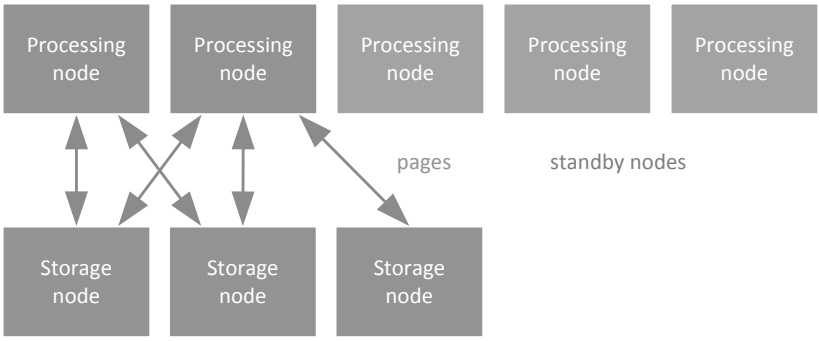


Figure 8.1: Elastic query processing on shared disk

8.1.2 Distributing queries

In our initial design, database objects are partitioned and distributed among a fixed number of nodes. To be able to determine the correct pages to access, the fixed partitioning scheme is known to all processing nodes. Further, to eliminate cache-coherence issues, we are running read-only OLAP workloads.

In this first implementation, a single node (*master node*) accepts queries from DB clients and selects a node to forward the query to. This node processes the query entirely, by fetching appropriate pages, reading records, and running necessary operations (sorting, grouping, aggregation).

The master node monitors utilization of all processing nodes and distributes incoming queries to the node with the lowest load. In case all processing nodes are running at full utilization, additional machines are powered up to assist in query processing. Likewise, underutilized nodes are excluded from receiving new queries, and can be suspended, after the last query finishes.

Processing nodes do not require any setup times, immediately after starting up, they are ready to evaluate queries. Likewise, shutting down superfluous nodes is fast, because no data needs to be transferred to other nodes.

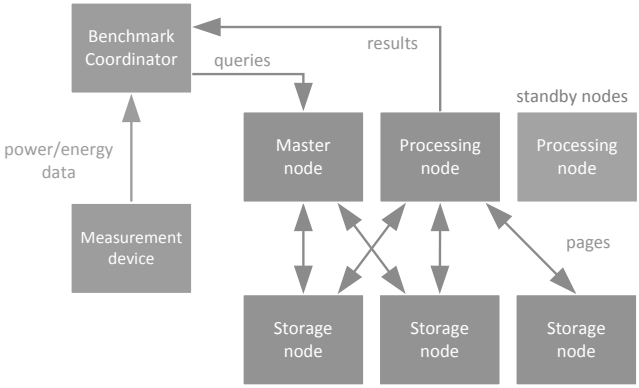


Figure 8.2: Experimental setup

Experiments

For these experiments, we are using an adapted TPC-H dataset[1] 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 have run a set of queries to test the abilities of WattDB to react to shifting workloads by distributing queries among processing nodes. In Figure 8.2, the experimental setup is depicted. Since query evaluation abilities of our implementation were limited at the time of these experiments, we have selected a subset of TPC-H queries—Q1 and Q4—to be run on the cluster.

While Q1 is an I/O-intensive query, sorting and aggregating a set of records, Q4 contains a *JOIN* operator and is therefore more computationally intense. Details about both queries, including SQL and query plans, are listed in Appendix A.2.

All queries are sent to the master node by a single computer, coordinating the benchmark and correlating query response times with power and energy figures from the measurement device.

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

storage nodes	processing nodes	response time [sec]	power [watts]	EDP [Js]
1	1	110.13	72.6	880,538.00
1	3	71.12	119.2	602,920.00
1	5	60.44	165.9	606,032.00
1	7	55.41	210.6	646,598.00
1	9	55.13	254.6	773,810.00
3	1	80.45	130.1	842,034.00
3	3	52.55	176.7	487,957.00
3	5	46.33	223.1	478,877.00
3	7	45	269.3	545,333.00
5	1	69.78	188.5	917,853.00
5	3	40.39	235.7	384,510.00
5	5	38.21	278.8	407,049.00
7	1	64.09	245.2	1,007,166.00
7	3	39.23	293	450,925.00
9	1	64.47	302.8	1,258,552.00

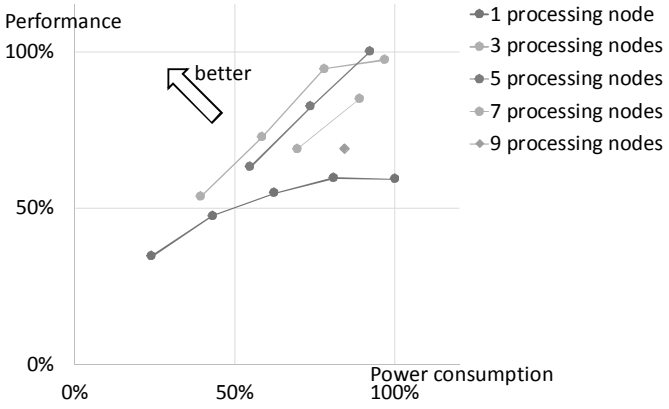
Table 8.1: Experimental results from distributing queries

On the storage nodes—equipped with 4 HDDs and 1 SSD—, we have generated the TPC-H dataset with a scale factor of 100, occupying approximately 200 GB (with additional indexes and storage overhead).

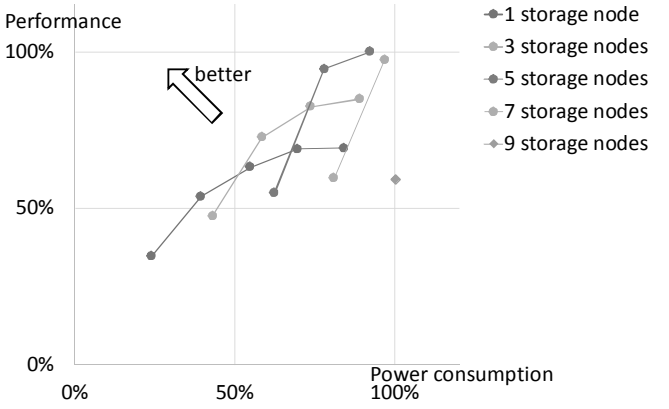
The benchmark client is submitting 10 streams of mixed queries (Q1 and Q4) in parallel to the database. Once a query result is returned, the next query is being submitted without delay. Hence, the cluster will have to process a highly concurrent workload.

Results

We have run the same workload on various, statically defined cluster sizes to estimate power/performance tradeoffs. Table 8.1 summarizes the average runtime and power consumption for all cluster sizes. As the results show, performance and power consumption increase with the number of nodes in the system. Yet, the right balance between storage and processing nodes is necessary to get optimal results. Running a large



(a) Processing nodes



(b) Storage nodes

Figure 8.3: **Energy and performance of distributing queries.** Both graphs show the same results; in the upper chart, data points with the same number of processing nodes are connected, while in the lower chart, results with the same number of storage nodes are lined up.

number of processing nodes on top of few storage nodes does not boost performance and, hence, is a waste of energy. Likewise, overprovisioning the storage layer to serve a thin processing layer is equally futile.

In the smallest configuration, consisting of a single storage node and one processing node (also acting as the master node), the cluster consumes approximately 73 Watts, yet, the query runtime is very high (110 seconds). The best performing configuration is made of 5 storage nodes and 5 processing nodes, delivering queries in 38 seconds, while consuming 279 watts. Replacing processing nodes with storage nodes results in higher power consumption without significant performance gains. Apparently, the 5-node storage layer is sufficient to supply pages to the processing layer.

To visualize the effects of scaling on both layers, Figures 8.3(a) and 8.3(b) plot performance and power consumption for all examined cluster sizes.

From these results, we conclude that it is possible to elastically scale the processing layer similar to the storage layer. Our experiments indicate, that read-only OLAP workloads seem to be good candidates to be processed on a shared-disk scale-out cluster of nodes.

Yet, query runtimes in our experiments were long and queries as the unit of distribution are rather coarse-grained. In preliminary tests, the master node had trouble determining underutilized nodes and placing queries accordingly.

Clearly, page servers come to their limits when dealing with complex queries. As we pointed out, it is inefficient to ship pages over the network and process them in remote nodes. Yet, the clear separation between processing and storage layer allows for easy reconfiguration and quick adaption to workloads without impacting data placement.

In [SH13a], we have further improved our DBMS to support query evaluation on storage nodes and enhanced the query evaluation power of WattDB to distribute query *operators* among nodes, instead of whole queries. While the storage nodes still primarily manage storage data, we have given them the ability to process simple query operators, i. e., projection and selection, on their underlying data, to preselect records for the processing layer. Further, we have enabled simple query processing capabilities on the storage nodes to mitigate the high latency of remote page shipping. Projection and selection operators are now running on the storage nodes, filtering the amount of records to send to

processing nodes. This little change turns the whole architecture from *page servers* to *record servers*.

Now instead of requesting pages from storage servers, processing nodes request sets of records, by pushing down database access operators to storage nodes. Hence, the bottom-most operators from the previous experiments, responsible for fetching records and for selection and projection processing, are now running in the storage nodes themselves.

8.1.3 Experiments

To verify performance, energy consumption and the overall elasticity of our approach, we have run a number of benchmarks on our record-server cluster.

Benchmarks on static clusters

First, we have repeated the previous tests, running TPC-H queries on statically configured clusters of various sizes. To test the influence of storage and processing on query performance and energy consumption, we have run the same workloads over and over again while varying the number of storage and processing nodes.

We generated the dataset on the storage nodes with a scale factor of 1 and provided indexes for the most important attributes. Data was distributed evenly among all storage nodes, hence, in case of a single storage node, all data was stored on hard disks attached to that node; and in case of 7 storage nodes, the larger tables (LINEITEM and ORDERS) were divided into equal-sized partitions and uniformly assigned among all storage nodes. The smaller tables (PART, SUPPLIER, PARTSUPP, CUSTOMER, NATION, REGION) are not partitioned and assigned to a single storage node. For our workload, we have selected TPC-H queries Q1 and Q4 as already described earlier.

After having generated the data, we evaluated a series of queries concurrently issued by a number of DB clients. Each test was running for 600 seconds.

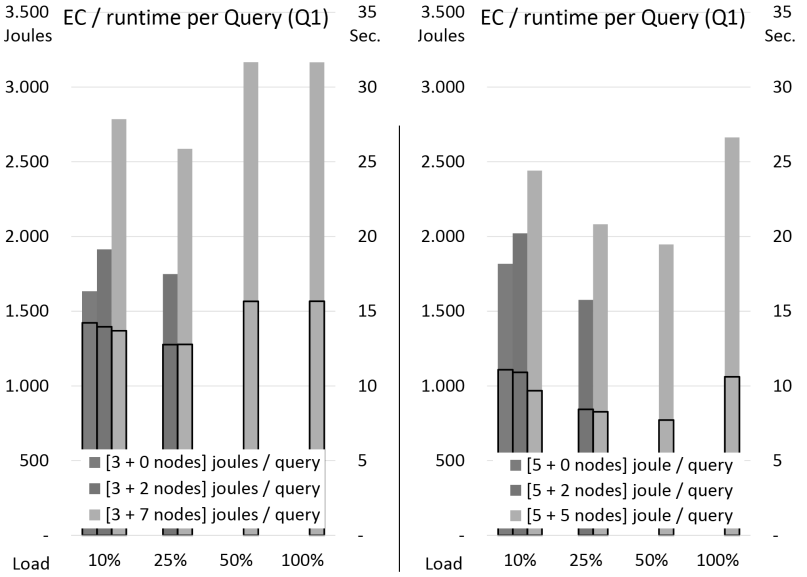


Figure 8.4: (a) Varying query load (from 10% to 100%) using TPC-H query Q1 on 3 storage nodes and 0 to 7 processing nodes. (b) Same query load on 5 storage nodes with up to 5 processing nodes.

Static cluster

First, we present measurements run on a fixed number of nodes to demonstrate that energy consumption and performance can be tuned to our needs. For this reason, we select the most energy-efficient configuration for each workload beforehand. To carve out the differences between the two queries, *Q1* and *Q4*, we have run them separately on the cluster. Each workload in our experiments has a differing number of parallel DB clients running on the benchmark/monitoring PC. The number of active DB clients, continuously sending queries to the database, is controlled by the benchmark specification. Thus, the utilization of the database changes with the amount of DB clients.

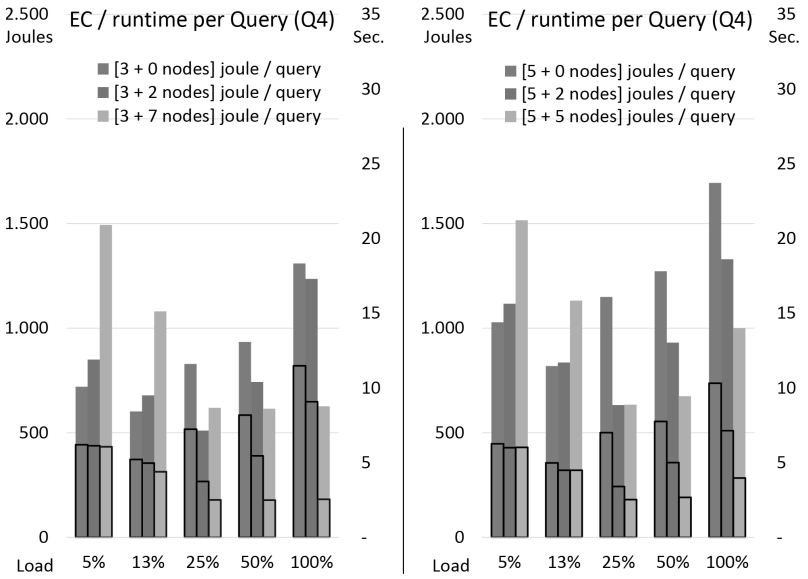


Figure 8.5: (a) Varying query load of TPC-H query Q4 using 3 storage nodes and up to 7 processing nodes. (b) Same query load on 5 storage nodes with 0 to 5 processing nodes.

Figures 8.4 (a) and (b) plot the energy consumption and performance running *Q1* on a cluster of nodes. Depending primarily on the number of nodes, the power consumption of a given cluster is more or less constant and, therefore, not shown in the graphs. The X-axis depicts the load of the cluster—starting with a very low load on the left and increasing it to the right. To characterize how the cluster is utilized, we define its load level x% by the number x of the DB clients. Hence, a load of 100% represents 100 DB clients. To enable better comparability, we normalized all results to *quantity per query*. The solid bars in the graphs illustrate the *energy consumption per query* (Y-axis on the left), whereas the framed (unfilled) bars report the *runtimes per query* (Y-axis on the right). The (up to) three bars (belonging to experiments characterizing the same utilization) represent clusters, where the number of nodes increase going from left to right.

We have evaluated identical workloads on different cluster sizes to demonstrate the dependency between energy consumption and performance. The smallest cluster, running the benchmark only on 3 storage nodes, has the lowest power consumption (∼114 W), but also the lowest performance (compare the left-most bars in Figure 8.4(a)). For low utilizations (see X-axis), the energy consumption per query is therefore minimal, although the runtime per query is higher, compared to larger cluster configurations. With an increasing number of DB clients, the small cluster comes to its limits and is unable to handle the workload, i. e., the available main memory is exhausted. Therefore, more powerful—yet more power-consuming—configurations take over. The next, larger configuration includes two more processing nodes (one of them is the already present master node). This configuration has a higher energy consumption under low utilization, but is able to process bigger workloads. Finally, we have combined 3 storage nodes with 7 processing nodes, creating an even more power-consuming configuration (∼246 W). Of course, its high number of active nodes leads to a waste of energy under low utilization, as the query processing time stays almost the same. The (3 +7 nodes) configuration is the only one powerful enough to handle all the workloads. Still, the query runtimes increased under high utilization, possibly due to a bottleneck in the I/O subsystem.

We have repeated the same benchmark on different cluster configurations having 5 storage nodes, depicted in Figure 8.4(b). Although the idle power consumption is higher and low-utilization workloads result in worse energy efficiency, the 5-storage-node cluster exhibits better energy efficiency under high loads. Increasing the number of processing nodes results in the same behavior as previously described, although the query runtimes at 50% load are further decreasing. On one hand, adding two more storage nodes instead of processing nodes and therefore increasing the I/O bandwidth pays off at high load. On the other hand, as the two additional nodes increase the lowest possible power consumption, energy efficiency at low utilization is worse. Therefore, we can already conclude that performance does not necessarily correlate with energy efficiency.

Next, we have executed the same benchmark for query *Q4*. The six cluster configurations were identical to the previous experiments. Figures 8.5 (a) and (b) illustrate the results for query *Q4*, where a utiliza-

tion of 100% represents 200 DB clients. The left figure depicts measurements on the cluster using 3 storage nodes ((3 + x) cluster), whereas the right figure shows those for the cluster with 5 storage nodes. All configurations are able to process the workloads, but with rising load, the query runtime gets worse. With a higher number of processing nodes, query performance is improved, whereas, however, energy consumption is increased. This result indicates that I/O bandwidth in the *Q4* experiments is sufficient. Now, the number of processing nodes is a performance-critical factor, as the cluster with (3 + 7) nodes exhibits a better performance than the 5-storage-node cluster, having only 5 processing nodes.

To further explore the influence of the I/O bandwidth in the cluster, we have measured the performance/energy outcome for the (1 + x)-nodes and (7 + x)-nodes cluster similar to the experiments in Figures 8.4 and 8.5. All results, including experiments on (1 + x)- and (7 + x)-node clusters can be found in Appendix A.3. In the (1 + x)-nodes cluster, bandwidth to disks is too low, thus, I/O latency for the individual tasks and, in turn, the entire processing times are strongly increased. As a consequence, query response times, throughput, and energy efficiency are impaired. Hence, we may not reach given performance goals, while we unnecessarily waste energy due to prolonged query runtimes. In the (7 + x)-nodes cluster, I/O bandwidth is not in short supply. Obviously, its static fraction of power consumption is higher, because storage nodes can't be turned off. As a result, the energy efficiency of this configuration at low utilization is worse, compared to smaller clusters. Although the 7 storage nodes provide enough disk bandwidth for all benchmarks, i. e., queries are not slowed down by I/O latencies, the overall energy efficiency suffers from the steadily high power consumption. Additionally, high workloads of *Q4* are afflicted with the limited availability of processing nodes.

These experiments clarify the importance of an adequate I/O subsystem. Because the two latter configurations (1 and 7 storage nodes) do not provide more insights, we have focused on the two middle-sized configurations presented here. For each query workload, an optimal configuration w.r.t. energy efficiency exists: The lower utilizations are handled best by the smallest cluster, as its performance is sufficient and it consumes the least power. With rising load, the next bigger cluster shows better energy efficiency. Finally, at full utilization, the most

powerful cluster offers the best efficiency, although it needs the most power.

These experiments on a statically configured cluster already reveal the opportunity of trading performance for energy savings. Yet, manually configuring the database to fit the expected workload is not optimal. For highly varying workloads, it is not even possible to select a single, fixed configuration with balanced performance and power consumption.

Hence, we have exposed the opportunity to trade performance for energy savings by manually selecting an adequate number of nodes to process the workloads. We have also shown that the best performing configuration is not always the most energy-efficient one. Instead, performance and energy efficiency are competing goals to be balanced. Static configurations require prior knowledge of the upcoming workloads and exhibit drawbacks under varying workloads, as a predefined configuration cannot exhibit ideal behavior for all load situations. By scaling the number of active nodes to the current need, we are able to dynamically adjust power consumption and performance. Therefore, we can select the best configuration, either in terms of power consumption, energy efficiency, or performance.

Dynamic cluster

After having evaluated the energy/performance behavior of static clusters, we wanted the cluster to dynamically adjust to a given workload, without needing predefined configurations. The cluster should tune the number of active processing nodes to fit the current load, and power up/down such nodes when the workload changes. To reach this goal, we use our power management component (*EnergyController*), running on the master node. This component is monitoring the current state of the cluster and is able to startup and shutdown (suspend) nodes. We already introduced and explained power management for storage nodes in [SH13b], where we experienced prolonged provisioning times due to the physical re-assignment of data segments and the resulting copy times. In this work, power management involves processing nodes, which do not have any additional startup cost and can come online in a matter of seconds. Similarly, suspending an underutilized processing node can occur immediately after finishing the last running query on it.

By running benchmarks on static configurations (see previous sec-

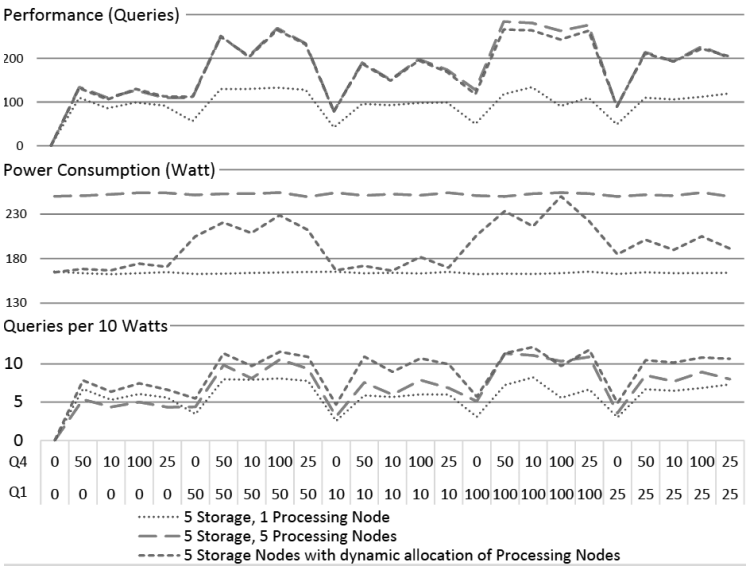


Figure 8.6: Experiments on three cluster configurations

tion), we have determined the maximum number of parallel queries per node and other limiting factors like CPU and memory bottlenecks. We are feeding this data to the *EnergyController* to improve the quality of its decisions, whether utilization in the cluster is evenly distributed.

After setting up the cluster, we are running a series of benchmarks with an increasing number of parallel queries of both, Q1 and Q4. In this experiment, we have run queries of both types in parallel, to generate a more complex workload with I/O- and CPU-intensive parts. Figure 8.6 plots the results in sequence. Each data point represents a number of DB clients running in parallel and sending either *Q1* or *Q4* queries to the cluster. The X-axis depicts the number of concurrent queries of each kind. The upper part of Figure 8.6 consists of three graphs, plotting *performance* as query throughput, *power consumption* of the cluster, and *energy efficiency* expressed in number of queries per 10 watts. To compare the measurements using a dynamically adjusting cluster, we have re-run the same benchmark on a fixed number of nodes, and also included the results in this graph. The short-dashed (blue) line repre-

sents the dynamic cluster, the long-dashed (red) line the fixed cluster with 5 processing nodes, and the dotted (black) line the (5 + 1)-nodes cluster.

The fixed cluster configurations mark the bounding box in which the dynamic cluster can adjust. While the small cluster shows the lowest power consumption, its query capabilities are limited. Therefore, it is unable to exceed a certain performance, even at high system utilization. The big cluster has the highest power consumption of all three configurations and, naturally, the highest performance. But under low system load, this performance potential lays waste and energy efficiency is impaired.

The dynamic cluster is able to adjust between both extremes and quickly adapts to the current workload. Therefore, its power consumption and performance covers the entire range between the two static configurations. Note, however, it more or less matches the performance of the big cluster, while it saves a substantial amount of energy. Gain in energy efficiency is particularly high for moderate cluster utilization, where maximum performance can be achieved with much less power consumption (see middle part of the benchmark in Figure 8.6). This last experiment proves that our cluster is capable of dynamically adjusting to a give workload. Yet, we have considered read-only OLAP queries here.

8.2 OLTP Workloads

In the previous experiments, we have run read-only workloads on a dynamically adjusting cluster. This restriction helped us understand the behavior of our system and allowed us to establish performance and energy-efficiency baselines. Yet, a typical DBMS also has to deal with creation, modification, and deletion of records. Further, transactions performing these operations still need to be ACID-compliant [HR83].

Introducing updates to a read-only system is a complex exercise, especially in a distributed environment. Not only do transactions need to be synchronized on each machine to prevent deadlocks, dirty reads, lost updates, and many more; global synchronization among all nodes is also necessary to detect wait-for cycles spanning multiple nodes and other anomalies [Rah94].

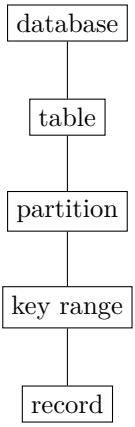


Figure 8.7: Lock hierarchy in WattDB

8.2.1 Implementation

In the previously introduced *record-server version* of WattDB, storage nodes are keeping the most recent version of records on disk. Naturally, storage nodes should thus be responsible for updating the data and making the changes persistent.

Our first implementation relies on a pessimistic, hierarchical lock protocol [Gra+76] to synchronize record accesses among transactions: Access granularities are (from small to big): Single records, B-tree ranges, partitions, tables, and (for metadata updates) the whole database as depicted in Figure 8.7. While smaller locks (record locks, B-tree range locks, and partition locks) are maintained on the respective storage nodes holding the data individually, the master node is responsible for synchronizing access to coarse-grained locks (table and database locks).

To synchronize individual wait-for graphs of each node, all machines exchange lock information periodically with the master node, where a global wait-for graph is maintained. Deadlocks are resolved by aborting and restarting the younger transaction, transparent to the database client.

8.2.2 Experiments

For our experiments with OLTP workloads, we have generated the TPC-C dataset with a scale factor of 100, resulting in approximately 10 GB of records stored on disk. With additional indexes and storage overhead, the dataset occupies 18 GB of data. Data is distributed among storage nodes using horizontal partitioning by Warehouse, as suggested by the TPC-C benchmark specification.

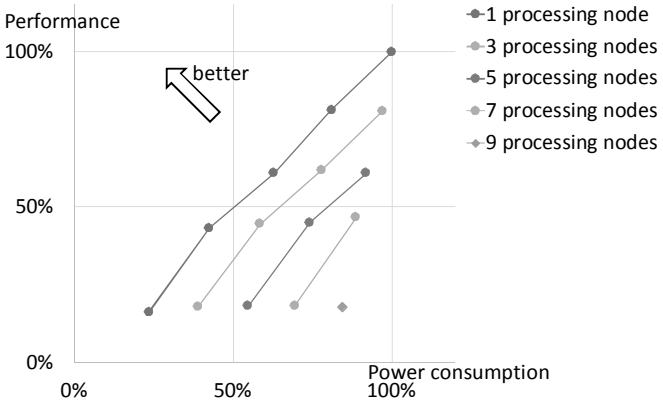
Because query evaluation capabilites of WattDB is still limited, we do not comply with the exact TPC-C benchmark specifications. To test the scalability and elasticity of our database, it was unnecessary to adhere to all requirements. For example, because our research prototype does not support multi-statement transactions with user interaction, we modified all queries to exclude (emulated) user interaction and to execute in "a single run" on the database. Further, our benchmark deviates from other specifications, e. g., wait time and response time constraints, 60-day space requirements, and transactions mix definitions.

We have set up a workload of multiple DB clients, all sending TPC-C queries to the master node, where requests are then routed to the processing/storage nodes. To test the scaleability of our cluster, we have set up numerous static configurations with varying numbers of storage and processing nodes. Each setup was benchmarked for performance in terms of query throughput and power consumption.

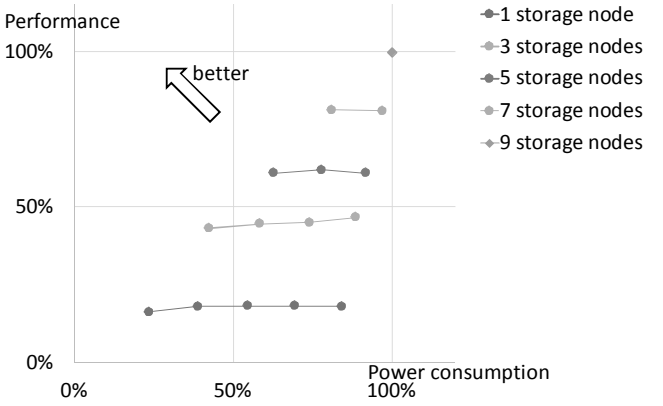
8.2.3 Results

Figure 8.8 plots performance and power consumption of all configurations from our experiments with OLAP workloads. In the upper subfigure, results are lined up by the same number of processing nodes, while, in the lower subfigure, results using the same number of storage nodes are connected.

While overall performance and power consumption increases with the number of storage nodes, surprisingly, the cluster does not benefit from adding processing nodes. This outcome is comprehensible, when considering the nature of TPC-C's OLTP queries: The majority of transactions is short-running and only touches a single Warehouse record. Typical operations running on processing nodes, like aggregations or grouping operators involving multi partitions, i. e., storage nodes, are



(a) Processing nodes



(b) Storage nodes

Figure 8.8: **Energy and Performance of distributing TPC-C query operators.** Both graphs show the same results; in the upper chart, data points with the same number of processing nodes are connected, while in the lower chart, results with the same number of storage nodes are lined up.

storage nodes	processing nodes	throughput [q / sec]	resp. time [sec]	power [watts]	EDP [Js]
1	1	4.27	2.34	71.59	39
1	3	4.72	2.33	118.58	59
1	5	4.76	2.31	166.2	81
1	7	4.76	2.31	210.52	102
1	9	4.74	2.32	255.53	125
3	1	11.39	2.37	128.99	27
3	3	11.76	2.38	177.52	36
3	5	11.86	2.36	225.07	45
3	7	12.29	2.36	269.44	52
5	1	16.06	2.74	190.72	33
5	3	16.36	2.75	236.73	40
5	5	16.06	2.74	279.02	48
7	1	21.45	2.89	245.93	33
7	3	21.38	2.9	295.05	40
9	1	26.4	3.03	303.73	35

Table 8.2: Experimental results from distributing OLTP operators

non-existent in TPC-C. Therefore, with this workload, there is no benefit in increasing the number of processing nodes.

Table 8.2 lists throughput, response times and power consumption in detail for each cluster configuration. As these results show, scaling-out on storage nodes (with query evaluation capabilities) slightly increases response times, due to the distribution overhead, but supports higher throughput, i. e., more parallel DB clients. Hence, a small cluster is best suited for lightweight utilization, whereas more intense, highly parallel workloads require larger clusters. The last column of Table 8.2 lists the Energy Delay Product for each configuration. To make a fair comparison, total energy consumption of the cluster is divided by the number of parallel queries. We observed comparable EDP results for all clusters with one processing node, hence, with the number of storage nodes, throughput increased linearly with energy consumption. Therefore, the cluster can be considered energy proportional.

8.3 Summary

The experiments conducted in this Chapter emphasize the importance of an adequate I/O-subsystem. As we have shown in Chapter 7, adapting the storage to changing workloads—although possible and energy-saving—is a cumbersome and slow task. Because of the time span needed to copy and move data, adjustments cannot be made every few seconds; hence, pre-selecting a robust storage configuration for the expected workload is essential. Changing the number of processing nodes as done in these experiments is a far more lightweight operation, as it does not require to restart queries or change data placement. WattDB is therefore able to react to workload variations within a few seconds.

By distributing queries among a dynamically adjusting number of nodes, energy consumption and performance can be tuned at a coarse-grained level. More sophisticated query operator placement strategies result in better node utilization and, thus, bigger energy savings. Also, with these experiments, we have shown that record servers are clearly superior to page-server-based clusters.

Yet, a cluster with separated storage and processing layers is inflexible and not very well fit for OLTP workloads. While OLAP queries can be spread over additional processing nodes, OLTP does not profit from a flexible processing layer, as query operators require close access to the data.

Combining both approaches (elastic storage and processing) to form a fully dynamic cluster is the logical next step to take.

9 Elastic DBMS

Our research indicates that elasticity is the key to an energy-proportional DBMS. Yet, previous experiments were focused on either the storage or processing layer and, thus, overall flexibility was limited.

To study the behaviour of a fully elastic DBMS, able to scale the number of storage nodes as well as the number of processing nodes, we have combined the concepts from both system versions in [SH14a]. In this work, we substantially extended the processing supported by WattDB to complex OLAP / OLTP workloads consisting of read-write transactions. For this purpose, we redefined and combined both approaches to get one step closer to a fully-featured DBMS.

Because the combination of (pre-)processing and storage on a single node deemed fruitful in our previous work, we treat all nodes identical now; hence, all nodes will act as storage and processing nodes simultaneously.

9.1 Architecture

In our previous solution (see Chapter 7), we migrated physical storage blocks among nodes to balance the cluster, and processing nodes had to request pages from storage nodes in order to process them. Hence, storage and processing were strictly separated. Later, we combined processing and storage (in Chapter 8), and dynamically adjusted the number of purely processing-centric nodes for elasticity. By introducing a *record server*, we have mitigated the latency issues of a *page server*.

Yet, in both approaches, data was exclusively assigned to nodes and transfer of ownership over data among nodes was not tackled.

To fully support elasticity, both the physical storage location *and* the logical ownership over data need to be dynamically assignable. Hence, instead of only moving storage blocks, logical data paths need to be flexibly adjusted.

Therefore, we have implemented a dynamic, logical partitioning scheme, that allows for flexible data migration. Additionally, the cluster is self-tuning to scale-out and -in according to the workload.

As in the previous implementation, 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 does not differ from the rest of the nodes; it is also able to process queries and manage its own storage disks.

9.1.1 Storage structures and indexes

In this implementation, 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 are implemented as B*-trees, by default, a primary-key index is always present. Additional indexes can be defined to speed up query processing.

9.1.2 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. Hence, we are employing *logical partitioning*, as described in 2.2.4.

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. 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.

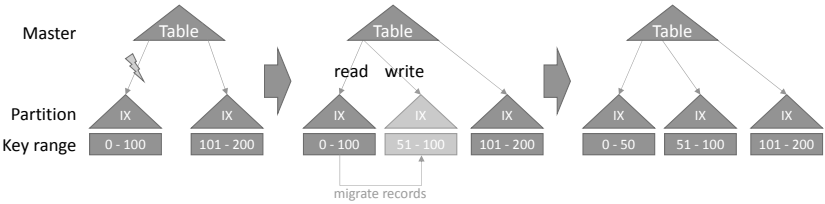


Figure 9.1: Three steps of a split in the partition tree

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 9.1 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 0 and 100 is split into two partitions, where a new partition has to be created. A transaction scans the old partition and moves records with keys between 51 and 100 to the new partition. Records with primary keys below 51 stay in the old partition.

While repartitioning is in progress, queries requesting records in the range from 51 to 100 need to scan both, the old and new partition, to ensure, all required records are read. Thus, reading transactions will access both, the new and old partition to look for records. The write pointer already references the new partition, redirecting all updates with primary key between 51 and 100 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.

9.1.3 Concurrency control

As every other DBMS, WattDB needs to implement mechanisms for concurrency control to ensure ACID properties [HR83]. 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.

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 [BG83]. 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).[1]

This property is especially useful for dynamic partitioning techniques, where records are frequently moved, i. e., deleted and inserted in another partition. In Figure 9.2, we have compared the performance of pessimistic locking (RX) with MVCC, while moving 50% of the records to another partition. To get an impression of MVCC's potential for our application, we have measured the performance and storage require-

[1] PostgreSQL calls it *vacuum*.

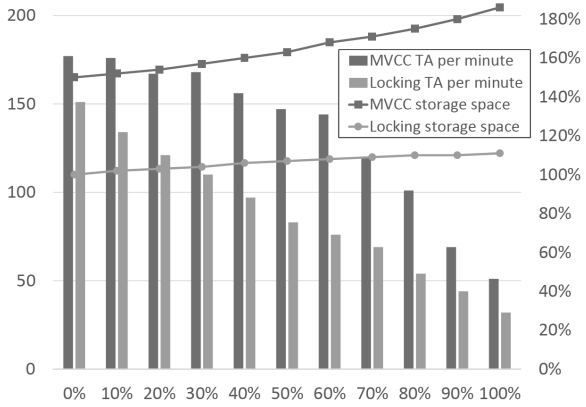


Figure 9.2: MVCC vs. locking: Performance and storage space consumption of different workloads while moving records

ments for different ratios of read-only and write-intensive transactions. The X-axis shows the percentage of update transactions, the remaining transactions are read-only. The graph bars depict the query throughput using MVCC and locking, respectively. The lines show the storage requirements for both mechanisms.

The experiment shows that MVCC can increase transaction throughput between 15% and almost 90%, while the affected partition is moved. Storage requirements for MVCC are naturally higher, as multiple versions of records have to be kept. Traditional locking also requires additional storage space to hold pending changes, which are applied to the data after the move finishes.

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 [Elm86]). 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.

9.1.4 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[2]
5. Update the partition tree information with final changes
6. Commit

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.

[2]Note that the order of Insert and Delete is not important.

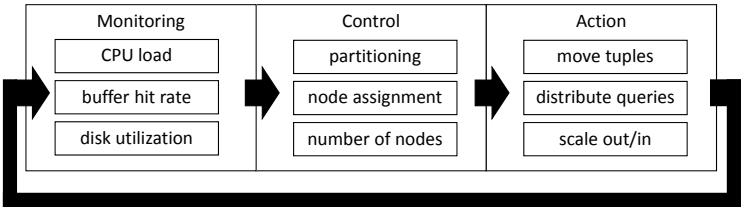


Figure 9.3: Monitoring & controlling the cluster

9.1.5 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.

9.1.6 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 9.3 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, CPU utilization, buffer hit rate, and disk utilization on each node is measured and sent to the master node. On the master node, the measurements are evalu-

ated, i. e., each performance indicator is compared to a predefined high and low threshold. If the master detects a 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 upper limit 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.

9.2 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, as described in detail in Section 3.7.4. 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.

We pre-created the TPC-H dataset with a scale factor of 100; hence, the DB holds 100 GB of raw data initially. 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. Altogether, a single *benchmark run* consists of 63 workloads, resulting in a total duration of ~2 hours.

9.3 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 mea-

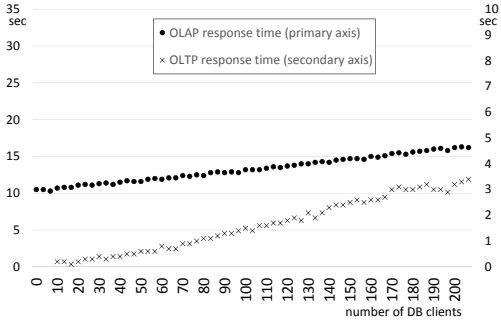
surements. Next, we ran the same benchmark against a fully dynamic cluster, where WattDB will adjust itself to fit the number of nodes and data distribution to the current workload ($BENCH_2$). Hence, initially all DB data was allocated to the disks of the master node. With growing number of DB clients, the dynamic partitioning scheme initiated a redistribution of the DB data with each additional node activated[3], 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 $BENCH_1$

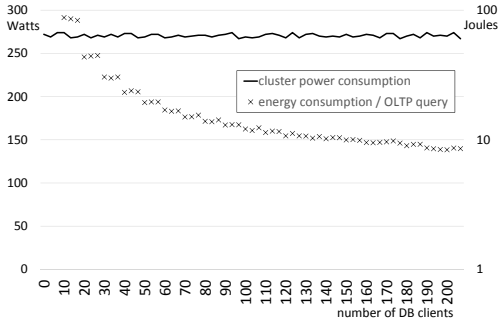
Figure 9.4(a) plots the performance of a static database cluster with 10 nodes. All nodes were constantly online and data was uniformly distributed on them. The number of DB clients is increasing over time (X-axis from left to right). Every client is sending OLTP queries sequentially, thus, the number 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.[4] Figure 9.4(a) 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 9.4(b) 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

[3] 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.

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



(a) Response time



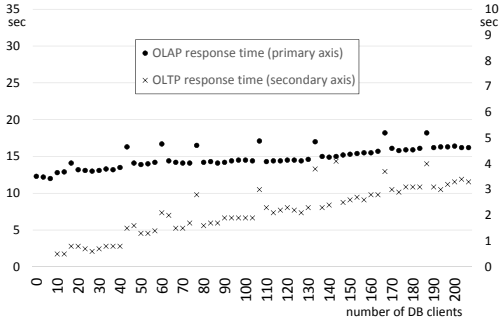
(b) Power/energy consumption

Figure 9.4: Increasing load on a static cluster

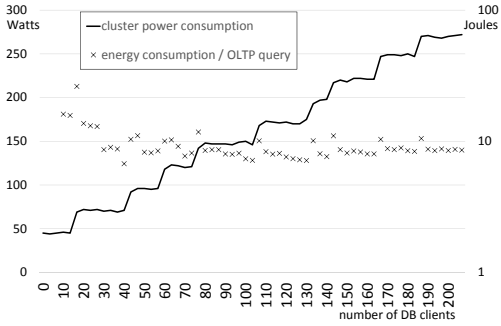
worse energy efficiency, especially at low utilization levels. As the plots indicate, energy consumption per query is very high at low utilization.

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 9.5(a) 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



(a) Response time



(b) Power/energy consumption

Figure 9.5: Increasing load on a dynamic cluster

consumption, as plotted in Figure 9.5(b), is initially low (about 45 W for the whole cluster). By calculating the integral over the differing courses of energy consumption in Figures 9.4(b) and 9.5(b), one can get an impression of the absolute energy saving possible (in joules), 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 9.5(a) 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.

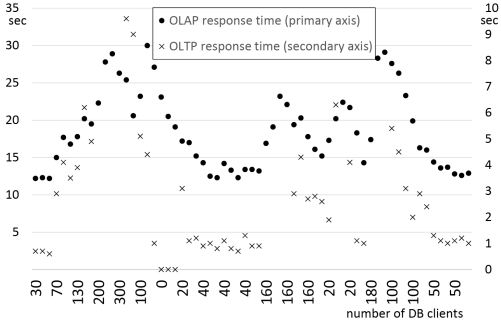
This experiment demonstrates that WattDB is able to dynamically adjust its configuration to varying needs. While minimizing the energy consumption at low utilization, the master node re-actively switches on more nodes as the workload rises. As a result, energy efficiency of the dynamic cluster is better 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.

Results of BENCH$_3$

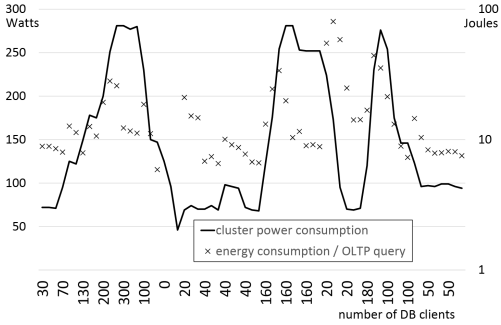
The previous experiments spawned an increasing number of DB clients to submit queries, providing a steadily increasing utilization of the cluster. Thus, the workload was rising slightly over time. Realistic benchmarks require more variance in load changes, in addition with quickly rising and falling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

In Figure 9.6(a), 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 9.6(b) 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



(a) Response time

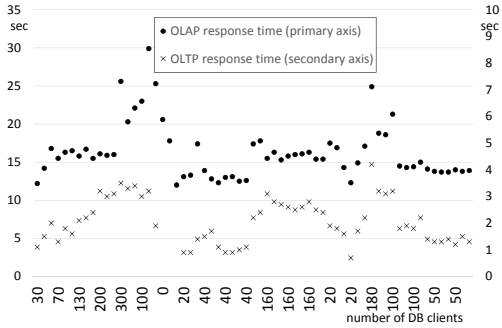


(b) Power/energy consumption

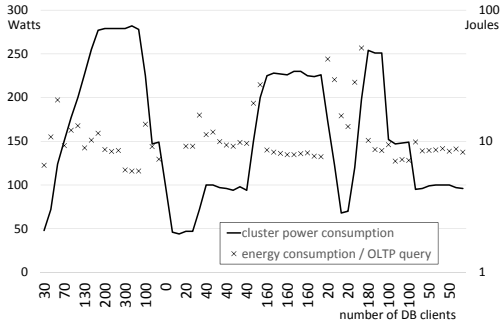
Figure 9.6: Varying load on a dynamic cluster

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.



(a) Response time



(b) Power/energy consumption

Figure 9.7: Varying load on a dynamic cluster supported by forecasting

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 by the master node to proactively partition data to match the worst-case demands of the current and the expected future workload. Figure 9.7(a) 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 9.6(a). On one hand, power and energy consumption, depicted in Figure 9.7(b), are higher under low utilization levels than on a purely reactive cluster (Figure 9.6(b)), 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.

9.4 Summary

After exploring opportunities for energy proportionality in the storage and query execution layer separately, we have combined both approaches in the previously presented experiments. 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 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. 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.

Therefore, a better suited partitioning scheme may better support elastic query processing than our initial implementation using MVCC.

9.5 Dynamic Aspects of Partitioning Revisited

Earlier, we have already explained the three different partitioning schemes (physical, logical and physiological partitionining). Here,

we give an overview of dynamic repartitioning aspects in all three approaches to motivate the need for *physiological* partitioning in our elastic DBMS.

9.5.1 Physical partitioning

Balancing under physical partitioning moves whole storage *segments* among nodes, without altering the data stored inside. Since the logical DB layer is oblivious of the segment distribution at the storage layer, logical access paths remain unchanged and keep pointing to the same (logical) page addresses while repartitioning, even when the physical placement of segments (and, thus, pages) changes.

A shared-disk architecture for the storage layer is needed to support physical partitioning among nodes, hence, every server needs to be able to access every segment, local and remote.

At the logical layer, all DB segments are exclusively assigned to nodes, independent of their disk placement to ensure integrity and eliminate the need for coordination. For this reason, cluster nodes will not share data stored in segments; regarding logical data accesses, it behaves like a shared-nothing architecture.

Because physical partitioning affects only the storage layer by distributing segments to disks/nodes, the query execution layer does not benefit from additional nodes hosting the data, because the logical control remains at the original node. Therefore, the query optimizer is unaware of the changes at the physical layer. Also, placing segments on remote nodes induces network access latencies, multitudes higher than local-disk access latencies. The intermediate network may also induce a bandwidth bottleneck.

Physical *re*partitioning does not require transactions, because logical records are not accessed; a lightweight latching/synchronization mechanism, locking segments on the move for a short time, is sufficient.

From an energy-concerned perspective, spreading data out to additional disks on remote nodes increases power consumption without adding much query performance.

9.5.2 Logical partitioning

In contrast to physical partitioning, *logical partitioning* moves records from one partition to another and, hence, affects the logical DB layer. To balance the workload among partitions, records within key ranges are moved between nodes. This requires the use of transactions to guarantee ACID properties: Records are removed from one partition and inserted into another; transactions need to ensure that concurrent transactions read either copy, but not both.

While rebalancing, dedicated transactions delete records in one partition and insert them into another. Hence, data movement alters the key ranges of the partitions. The query optimizer can take the new partition distribution into account for future query optimization.

Spreading data over multiple partitions, stored on separate nodes, reduces latency and increases bandwidth at the physical layer due to the increased number of disks—as with physical partitioning. Further, by logically dividing the data into key ranges, stored separately, the query optimizer can prune unneeded partitions. Additionally, using logical partitioning, ownership of the records changes and all nodes holding segments can access partitions in parallel and, thus, speed up query processing.

Yet, to remove records with a specific key range from a partition, a large part of the data must be read and updated, possibly scattered among physical pages. Hence, logical partitioning is more IO-heavy than physical partitioning. Since transactions are needed, queries running in parallel may get delayed due to locking conflicts.

9.5.3 Physiological partitioning

Since physiological partitioning encapsulates small groups of records into self-contained mini-partitions, each stored in a single segment), moving a segment from one partition to another does not invalidate the primary-key index of the segment.

To repartition a table, it is sufficient to move whole segments, containing mini-partitions, to another node. The receiving node can immediately resume query processing, while old transactions may still finish reading from the old segment on the sending node. New queries will already access the segment on the new node. To reflect changes in the

partitioned DB, only an update to both of the meta-indexes (of the new and old partition) is required.

Like physical partitioning, physiological partitioning copies data almost at raw-disk speed. Additionally, the logical layer is aware of the new data distribution and can participate in query processing as with logical partitioning, e. g., the query optimizer can perform *segment pruning*, allowing a query to quickly identify unnecessary segments, having no interesting data. Also, buffering, synchronization, and integrity control for the segment are now transferable to another node—not possible with physical partitioning. Using physiological partitioning, we can still apply MVCC for concurrency control.

Repartitioning details

Rebalancing the DB cluster, exemplified by the movement of a single segment, works as follows: First, the partition is marked for repartitioning at the master node and the partition tree at the source node is updated with a pointer to the new location of the partition. Next, at the source node, a read lock is acquired on the source partition, waiting for pre-existing queries to finish updating the partition. Updating transactions need to commit before the lock is granted. By ensuring that all changes to the partition are committed, no UNDO information needs to be shipped to another node. After the lock is granted, the partition is copied to the target node and inserted into the node's partition tree. At this point, the new partition is unlocked and records in it can be accessed by readers and writers again. The master node is informed of the successful movement operation, and the global partition table is updated accordingly. New transactions will now access the new node directly. The partition information at the source node still points to the target node, redirecting all queries trying to access the old partition to the new one. Finally, after all old transactions no longer want to access the old partition, the master informs the old node to unlock the partition. At that time, the pointer to the new node is removed from the source node and the old partition can safely be removed.

Logging

For durability reasons, write-ahead logs must be maintained at all times. When repartitioning, although record ownership changes, log files remain at the original node and are not transferred to the node hosting the partition. In case of DB failures, the log file is needed to reconstruct partitions and to perform appropriate UNDO and REDO operations. Since moving a partition involves read-locking the entire partition, this operation acts as a checkpoint. All transactions before the movement will have their actions recorded in the old log file. While moving the partition, copies of the records still remain until the movement is finished. Hence, additional logging is not required. After successfully moving a partition to another node, the partition will be in a consistent state and flushed to disk. Hence, the old copies and log files are no longer required. Now, updates to the partition can be logged at the new node.

Housekeeping at the master

Query optimization is done at the master node. To identify all partitions relevant to a query, the master keeps a tree with the primary-key ranges of all partitions. While repartitioning, both the sending and receiving node need to be accessed by queries to determine which node currently claims ownership over the data. Therefore, when repartitioning starts, the master is updated first, keeping pointers to both, the old and new node. After repartitioning, the old pointer is deleted.

Correctness

Dynamic data migration must not alter the result of concurrent queries; therefore, ACID properties must be maintained at all times. Due to the copying/moving of records among partitions, transactions may access both copies or none of the copies by mistake. In contrast to logical partitioning, rebalancing physiological partitions requires some modifications to MVCC to ensure correctness. To show that transactions will behave correctly, we must provide a proof of correctness for transactions at different starting times and distinguish between reading and writing accesses.

First, **transactions started prior to rebalancing** must be able to access old versions of the records. Since the copies are kept until

all old readers are finished, these transactions will always be able to read. During rebalancing, a read lock is acquired on the old partition, ensuring that all writing transactions will finish until the partition is moved. Newer transactions, arriving after locking the old partition, are either arriving at the new location and may write immediately, or are forced to wait for the copying to finish and are then redirected to the new partition.

Second, **transactions started after rebalancing** must not access old copies. After updating the partition tree at the master node, new transactions will be redirected automatically to the new partition. Before the update, transactions will behave identical to the first case and potential updates will be redirected to the new partition, where proper synchronization is enforced.

There is a small time window, where the partition tree on the master is not up to date and transactions may get routed to the old partition instead of the new one. Therefore, partitions on the move have overlapping primary-key ranges on the master (as depicted in Figure 2.16) and queries are advised to visit both, determining the correct location to use during execution.

Reading transactions will have to touch both partitions, potentially reading the same record twice. Yet, MVCC will ensure, that only one version of the record is deemed eligible for the query, while the second one is silently discarded. Likewise, writing transactions will only overwrite the newer version of the record.

Therefore, we conclude that moving partitions works correctly if the utilized MVCC protocol is correct, which—despite programming errors—was already proven in the original publication on multiversion concurrency control [BG83].

9.6 Experiments

To compare energy consumption and performance impact of various partitioning schemes, we have evaluated all three implementations on our cluster with an OLTP workload in [SH15]. In the following, we first describe the experimental setup, before we present results from our work.

9.6.1 Experimental setup

For all experiments, we are using the dataset from the well-known TPC-C benchmark with a scale factor of 1,000. Hence, a thousand warehouses were generated on the cluster, consisting of about 100 GB of data. Due to additional indexes and storage overhead, the final DB had approx. 200 GByte of raw data.

Queries

We use queries from the TPC-C benchmark as workload drivers for our experiment. Because we do not compare our results with other TPC-C results, we do not comply with the exact TPC-C benchmark specifications which are unessential to reveal differences of partitioning schemes, as already mentioned in Section 8.2.2.

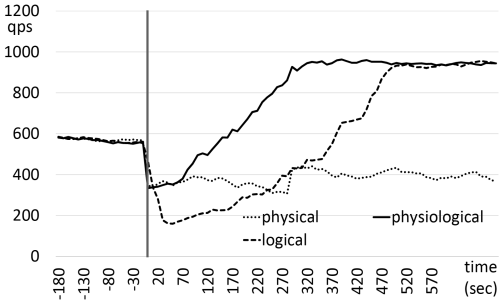
Workload mix

In each experiment, we spawned a number of OLTP clients, sending queries to the DBMS. Each client submits a randomly selected query at specified intervals. If the query is answered, the next query is delayed until the subsequent interval similar to defined think times in the TPC-C specification. Hence, the more OLTP clients and the lower the think time, the more utilization is generated.

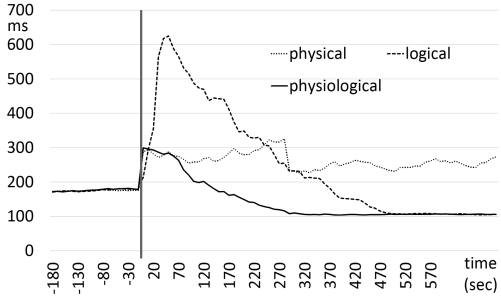
By limiting the maximum throughput at the client side, this experiment differs from traditional benchmarking. While established benchmarks such as TPC-C use maximum throughput as the metric, we are interested in the DBMS fitness to adjust to a given workload by keeping throughput acceptable and optimize the number of nodes the DBMS is running on, and, thus, improve energy efficiency.

Partitioning

As previously described, the limiting factor for dynamic repartitioning is migration cost, i. e., the performance impact and time to move data among nodes. To estimate its impact on the cluster's elasticity, we have conducted a simple experiment: Starting with two nodes, hosting the data and processing queries, we instruct WattDB to perform a repartitioning of all tables and migrate 50% of the records to two additional



(a) Throughput of the cluster



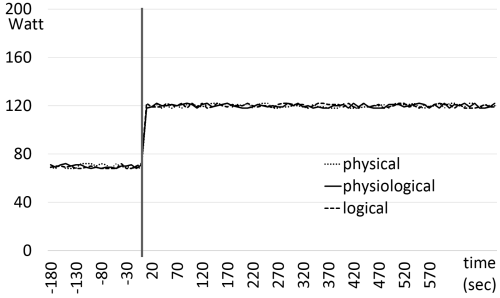
(b) Avg. response time per query

Figure 9.8: Performance-related benchmark results for different partitioning schemes under a TPC-C query mix

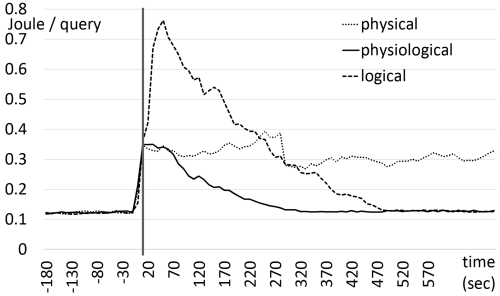
nodes. We measure response time, throughput, and power consumption of the cluster before, during, and after the repartitioning. We repeated the experiment on all three types of partitioning schemes, physical, logical, and physiological and compared the results.

9.6.2 Results

In this section, we present results from our experiments on clusters using logical, physical, and physiological clustering. Figures 9.8 and 9.9 illustrate how—under the experiment sketched—query throughput, response time, power consumption, and energy use per query for the three



(a) Power consumption of the cluster



(b) Energy consumption per query

Figure 9.9: Energy-related benchmark results for different partitioning schemes under a TPC-C query mix

benchmark runs evolve over time. In each graph, the X-axis indicates the time measured since initiating rebalancing in seconds. At time $t \pm 0$, the cluster was instructed to rebalance as previously described. For $t < 0$, the results are more or less identical, because the initial configurations were identical for all experiments. After starting repartitioning, measurements begin to differ based on the selected partitioning scheme. Because the same number of machines was used, power consumption is almost identical in all cases.

Physical partitioning

Immediately after initiating rebalancing, query response time slightly increases and throughput reduces from about 600 to 400 qps, due to the network overhead of copying segments and the increased latency to access the now remote pages. After moving 50% of segments to new nodes (around $t + 270$), query response time decreases, but does not recover to its old level.

With physical partitioning, segments are moved to another node, but are still "owned" by the original node. Therefore, the partition can benefit from higher IOPS, due to the distribution, but suffers from increased network latency, because segment access now requires a remote call.

Referring to the measurements, we reason that physical partitioning—although easy to implement—is not usable for a dynamic cluster of DBMS nodes. Applying this technique, we can distribute data among multiple disks, but the logical control of the data is stuck at the original node. For this reason, storage segments have to be fetched from that node to access their records, which imposes additional latency. Furthermore, without additional CPUs and main memory to help evaluating queries, scale-out can only be achieved at the storage layer. Thus, physical partitioning is not useful for a fully dynamic DB cluster. This observation is consistent with our experiments on the physical storage layer in Chapter 7.

Logical partitioning

Using logical partitioning, the control over a key range—together with the records—is transferred to another node. Hence, moving records of a key range $[a - b)$ to another node requires the node to evaluate queries for that key range from now on.

The benchmark results on a logically partitioned cluster indicate an initial decline in query throughput (Figure 9.8(a), at $t \pm 0$). Compared to the other schemes, logical partitioning exhibits the highest query response times when rebalancing (Figure 9.8(b)). After a significant amount of records has been relocated to other nodes, throughput and response times start to improve (at t+170) and quickly pass performance before repartitioning.

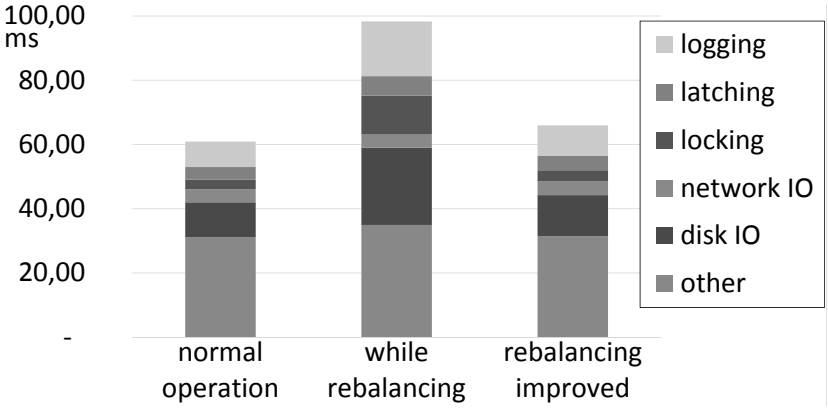


Figure 9.10: Impact factors on query runtime when rebalancing

We explain the initial performance setback with the additional high system load due to table scan(s) and the network load for finding and moving records. With parts of the data moved to another node, the original node does only have to manage the remainder of the data and the additional node takes part in query processing, doubling the number of CPUs and main memory available.

Hence, with logical partitioning, it is possible to add storage and processing power to the system, making it a better candidate for a dynamically adjusting cluster. Yet, moving key ranges and scanning for data is time-consuming, compared to raw movement of physical segments.

Physiological partitioning

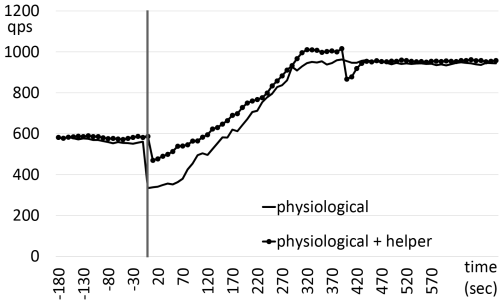
The corresponding results for our benchmark on a physiologically partitioned cluster exhibit an initial decline in query performance (throughput and response times) similar to physical partitioning. Similar to logical partitioning, performance quickly recovers and soon outperforms physical partitioning (around $t + 250$), as soon as the majority of segments is transferred to the new nodes. At this point, response times start to get lower than before, because all nodes can now participate in query processing.

In our experiments, physiological partitioning exhibited the lowest query runtimes and handles repartitioning events well, compared to the other approaches. It provides fast adaptation of data partitioning in a dynamic cluster and quickly compensates data migration overhead. Overall, physiological partitioning delivers best energy efficiency and quickest adaptivity. With this approach, we are combining the speed of data movement with the ability of transferring ownership of data. The DBMS moves segments among nodes at the same speed as with physical partitioning. As soon as segments arrive at the new node, they are incorporated in its index and the new node overtakes query processing. Yet, immediately after the beginning of repartitioning, performance declines, further slowing down query processing. Therefore, physiological partitioning still shows drawbacks that need to be tackled.

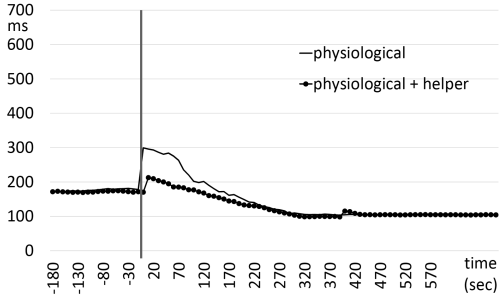
Physiological partitioning improved

From our first experiment on a cluster using physiological partitioning, we experienced slow query response times during repartitioning. We analyzed the performance setback and identified bottlenecks in the cluster. In Figure 9.10, the major impact factors on query runtime are illustrated for a physiologically partitioned cluster. On the left side, the graph shows a breakdown of time spent in various DBMS components when running queries. On the right side, the same queries are running while the data is rebalanced to other nodes. From the increase in runtimes, we can deduce that critical sections are disk I/O and locking. Surprisingly, although repartitioning ships big chunks of data across the network, the time spent for network communication remains unchanged. The findings indicate several bottlenecks: First, locking partitions keeps queries waiting and, thus, increases runtime. In our implementation of the rebalancing operation, the lock is essential for data integrity. Therefore, there is nothing we can do to mitigate locking overhead.

Second, rebalancing involves heavy I/O, competing with disk accesses of regular queries. Reducing accesses to hard disk would therefore speed up query processing while repartitioning. Additionally, we noticed more contention in the DB buffer due to a pile of waiting queries with latched pages and occupied pages needed for rebalancing (not shown in the figure). More DRAM might reduce page thrashing and relieve the storage subsystem.



(a) Throughput of the cluster



(b) Avg. response time per query

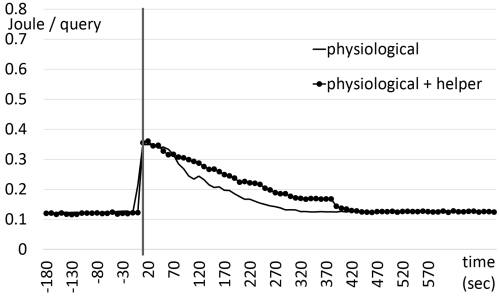
Figure 9.11: Improving the benchmark results for physiological partitioning; performance and throughput

Lastly, as shown in Figure 9.10, logging takes significantly longer when rebalancing. Since logging writes to disk as well, we conclude that the main bottleneck for repartitioning seems to be the bandwidth to the storage subsystem.

To mitigate excessive load on the cluster while rebalancing, we conducted a final experiment, where we powered up additional nodes to assist the present ones. Since offloading OLTP query operators to remote nodes is not reasonable, we used the helper nodes for log shipping and provision of additional buffer space using rDMA[5]. Accessing buffer

[5]rDMA = remote direct memory access. A node's buffer size is increased by including main memory from remote nodes.

(a) Power consumption of the cluster



(b) Energy consumption per query

Figure 9.12: Improving the benchmark results for physiological partitioning; energy and power

pages from a remote memory includes network latency, but is still faster than flushing a page from the buffer and reading it back from disk when needed. Especially *warm* data, that is not accessed frequently in the buffer (but frequently enough to justify keeping the page in memory), is a good candidate for rDMA buffering. The graphs in Figures 9.11 and 9.12 plot the results in comparison with "standard" physiological partitioning. At time $t \pm 0$, when repartitioning started, two additional nodes were fired up to support the cluster. After repartitioning was finished, the helper nodes were brought down again (around time $t + 370$). As the results confirm, including additional nodes increases power consumption (Figure 9.12(b)), but improves query response times

(Figure 9.11(b)). Overall, energy efficiency gets worse (more energy consumption per query, see Figure 9.12(b)), but, in turn, performance increases (Figure 9.11(a)).

We conclude that adding nodes during rebalancing helps mitigate data shipment overhead at the cost of higher power consumption. Therefore, after rebalancing, the additional nodes should be turned off again to improve energy efficiency of the cluster.

9.7 Summary

With these experiments, we have evaluated different partitioning approaches to be used in a distributed DBMS, which approximates energy proportionality by dynamically adjusting the cluster size to the workload present. Our experiments identified drawbacks with physical and logical partitioning schemes and recommended physiological partitioning as best choice for dynamically repartitioning a database under load. Physical partitioning is the easiest to implement and offers quick repartitioning. Yet, due to its physical nature, query processing does not benefit from rebalancing, which is a huge drawback in an elastic environment.

Logical partitioning seems to be the cleanest way of balancing, as it changes logical key ranges and their assignment to nodes. Yet, a straightforward implementation suffers from high repartitioning cost which makes dynamic adaption time intensive.

Finally, our proposed physiological repartitioning, a combination of both approaches, offers short rebalancing times with the benefit of changing logical data ownership. Hence, data and processing can be dynamically redistributed among a cluster of nodes, allowing to form an elastic, energy-proportional cluster.

Yet, the experiments indicated that repartitioning an already stressed cluster imposes additional load and slows down query evaluation. By activating additional nodes to support query processing, we were able to relieve some of the stress and to improve responsiveness of the system. In conclusion, we were able to trade energy efficiency for performance.

10 Comparison with a Brawny Server

After exploring and implementing techniques to form an elastic, energy-proportional cluster of nodes, the question remains, whether the system is comparable to a centralized, heavy-weight server (also called *big server*).

The comparison of a cluster and a big server is threefold. First, we provide a lineup of two theoretical systems and compare their performance, energy efficiency, and total cost. Next, we run benchmarks on our cluster and another, centralized system to get actual performance and energy consumption figures. Finally, we propose a system to combine the advantages of both approaches.

10.1 Total Cost-of-Ownership Analysis

In [Sch11], Scherer analyzed the total cost of ownership for two hardware platforms: A single, heavy-weight machine and a cluster of ten nodes, capable of individually powering on and suspending to match the workload. Both systems were tested with the same workload traces, provided by SPH AG, a mid-sized ERP-developer [SHK12].

10.1.1 TCO model

The term *Total Cost of Ownership* (TCO) was developed by Gartner in 1987 for its customer Microsoft, to compare Microsoft-based infrastructures to Linux-based ones [MK05]. They defined TCO as follows: "A comprehensive assessment of information technology (IT) or other costs across enterprise boundaries over time. For IT, TCO includes hardware and software acquisition, management and support, communications,

end-user expenses, and the opportunity cost of downtime, training. and other productivity losses" [Inc12].

Koomey, Brill, Turner, Stanley, and Taylor extended Gartner's TCO approach in [Koo+07], to tailor calculation for datacenters. In their approach, called *True TCO*, irrelevant costs were excluded, e. g., software and license acquisition costs, and, instead, energy-related cost factors were included. They distinguish between *Capital Expenses*, i. e., one-time investments for hardware acquisition, and *Operational Expenses*, which are recurring spendings for running costs, e. g., energy.

Today, there exist many different definitions of TCO—basically each consulting company developed their own. Therefore, we need to clarify the model used and define all cost factors that are included. In his work, Scherer adapts the concept of Koomey's *True TCO* and simplifies it to be able to quickly compare the two platforms. While *True TCO* includes estate costs and taxes—factors which are relevant for datacenters—, Scherer derives costs estimates from server housing prices. He argues that rental prices include proportions of all relevant costs, e. g., maintenance, housing, cabling, etc, plus a profit margin and, therefore, justify a square comparisons, despite the simplification. Additionally, Scherer compares only two dedicated platforms and results are not intended to be used for datacenter comparison.

Similar to the other TCO approaches, to distribute capital expenses over the lifetime, they are converted into annual write-offs. Hence, acquisition costs for servers, networking, and storage are split among the years of their supposed life cycle.

10.1.2 Hardware selection

Comparing two hardware platforms is not as trivial as it may seem. There are several requirements for two platforms to be considered comparable. Both systems should exhibit similar performance figures, e. g., processor, disk, and main memory (at least) should be comparable. Especially two different architectures, i. e., a centralized system and a distributed cluster, exhibit very different performance characteristics. As Lang, Patel, and Shankar have demonstrated in [LPS10], clusters suffer from friction losses, further complicating the selection of a proper big server.

	Cluster	Big Server
# of processors	10	2
Processor type	Intel Atom D510	Intel Xeon L5630
# of cores	10 x 2	2 x 4
# of threads	10 x 4	2 x 8
Frequency	1,66 GHz	2,13 GHz
Cache	10 x 1 MB	2 x 12 MB
Main memory	10 x 2 GB DRAM	24 GB DRAM
Power consumption idle	1 x 24,425 watts	110,9 watts
Power consumption peak	10 x 29,825 watts	221,8 watts
Switch	HP 1810G-24	(not needed)
Hard disks	10 x WD Scorpio Black (250 GB)	

Table 10.1: Hardware components, according to [Sch11]

Cluster

Since development on WattDB started prior to the work of Scherer, the hardware for the cluster was already selected. Table 10.1 gives a quick overview of the components, while the details were already explained in Section 5.2.

Big server

Based on the cluster hardware, a centralized system was needed, that delivered comparable performance. To ensure technological comparability, the system should have been rolled out to market at the same time as the cluster hardware.

Scherer selected a server with two Intel Xeon L5630 processors and 24 GB of DRAM for comparison. While these CPUs offer higher frequencies (2.33 GHz, instead of 1.66 GHz), the total number of threads is reduced from 40 in the cluster to 16 in the big server. The big server scores about 12 times higher in the *Passmark* benchmark, hence, performance should be comparable to the cluster, if not slightly higher.

Both platforms share the same external storage configuration, consisting of ten hard disk drives. Since the distributed nodes require an additional communication infrastructure, Scherer included the network-

ing switch into the TCO calculation of the cluster. The power consumption of the big server rougly equals the plot from Figure 2.12, i. e., at 20% utilization, the server already consumes 80% of its peak power.

10.1.3 Workload profile

To calculate the TCO of the systems, typical workload patterns are required to estimate average energy consumption of the lifetime. SPH AG provided CPU and disk utilization patterns as described in Section 2.1.2, which Scherer used as input traces to estimate performance, reaction times, and power consumption of both systems. By correlating workloads with the expected server utilization, power consumption can be estimated.

10.1.4 Scale-out policy

The cluster is designed to elastically scale-out and -in, based on the current workload. Since at the time of Scherer's publication, WattDB was not yet ready to automatically adapt. Therefore, policies defining reaction thresholds and reconfiguration times are needed to simulate the cluster's behavior.

Scherer developed a policy to model the behavior of WattDB under variable workloads. He assumed, the master node checks the current utilization every 10 seconds and in case imbalances are discovered, rebalancing is triggered. Reconfiguration times to power on an additional node were set to 60 seconds, while time to shutting down was defined to take 10 seconds. Workloads that were not processed in one period were piling up and passed to the next one.

To grasp the differences between energy saving and high-throughput policies, Scherer varied reaction and scheduling times on the master to either eagerly switch on additional nodes or to reduce overall power consumption during workload spikes.

To mitigate the effect of sudden workload shifts, scale-out is triggered manually, before the workloads hit the cluster. In reality, these manual triggers can be replaced by appropriate forecasting, as discussed in Section 5.5.

	OLTP		OLAP	
	Big Server	Cluster	Big Server	Cluster
Capital Expenses				
Server hardware	640 €	654 €	640 €	654 €
Disk drives				104.87 €
Network switch		88 €		88 €
Operational Expenses				
Housing	72 €	264 €	72 €	264 €
Electricity	211 €	156 €	239 €	82 €
Cooling	211 €	156 €	239 €	82 €
Total	**1,239 €**	**1,422 €**	**1,296 €**	**1,273 €**

Table 10.2: TCO comparison at 0.20 € / kWh electricity, from [Sch11]

10.1.5 Results

Scherer calculated the TCO for both systems using workloads from SPH AG and his simulation policies, depicted in Table 10.2. He discovered that cluster and big server deliver comparable costs, yet overall, the big server's TCO might be a little lower. Especially for OLAP workloads, the cluster's TCO is 11% lower, which indicates potential to save costs when replacing a single OLAP server with a cluster of nodes.

His work was a pessimistic analysis of both architectures, including high friction losses for distributing operations and moderately low energy prices. The results provided in this simulation provided first approaches to compare the performance of both platforms and indicate areas, where the cluster might be better suited than the big server. Yet, as Scherer concludes, switching to a distributed architecture is only possible if potential delays in queries—due to reconfiguration overhead—are bearable.

In his work, Scherer proposes forecasting as a viable mechanism to prepare the cluster for upcoming workloads. Further, he proposed heterogeneous architectures, combining both approaches, e. g., to process OLTP workloads on a big server while offloading analytical queries to an elastic cluster. Although Scherer's TCO analysis was based on many assumptions and not backed with benchmarks, his findings are conclusive to our experimental results.

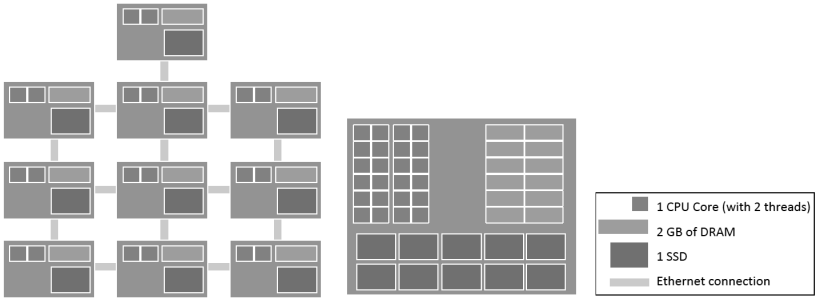


Figure 10.1: The 10-node cluster compared with the brawny server

10.2 Experimental Examination

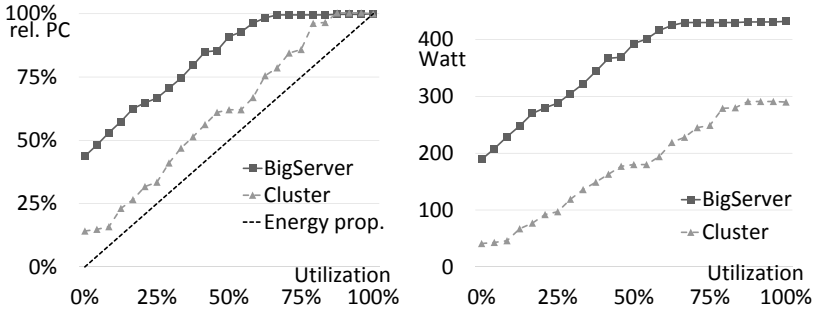
After analyzing the efficiency and total costs of a dynamic cluster by simulation and estimation, we have implemented WattDB and ran comparison benchmarks on the cluster and a big server to verify the results provided in [Sch11]. In [SH14b], we have published our results of an experimental comparison of both platforms.

We compared a single, brawny server with a cluster of wimpy nodes under OLTP and OLAP workloads, running TPC-H and TPC-C, respectively. We ran several empirical experiments and compared energy use and performance of our cluster to those of a brawny server.

10.2.1 Hardware

Our cluster hardware consists of the 10 well-known nodes, interconnected by a Gigabit-ethernet switch. Each node is equipped with an Intel Atom D510 CPU (with two threads using HyperThreading) running at 1.66 GHz, 2 GB of DRAM and an SSD for data storage. All nodes can communicate directly.

To compare performance and energy savings, we ran the same experiments again on a single, brawny server. This server has two *Intel Xeon X5670* processors with 24 GB of RAM and 10 SSDs. For a fair comparison with the wimpy nodes, we have reduced the RAM to 24GB, although the server can handle much more. Yet, with more main memory, the power consumption of the server would also be much higher.



(a) Relative power consumption (b) Absolute power consumtion

Figure 10.2: Power consumption for both systems

Each CPU has 12 cores and 24 threads (using HyperThreading), running at 2.93 GHz.

Figure 10.1 sketches the cluster with 10 nodes and the big server. For comparison, we have highlighted the main components (CPU cores, main memory and disk) inside the nodes as well as the communication network.

Each wimpy node consumes ~22 – 26 watts when active (based on utilization) and ~2.5 watts in standby. The interconnecting network switch consumes ~20 watts and is included in all measurements.

In its minimal configuration—with only one node and the switch running and all other nodes in standby—the cluster consumes approx. 65 watts. This configuration does not include any disk drives, hence, a more realistic minimal configuration requires about 70 watts. In this state, a single node is serving the entire DBMS functionality (storage, processing, and cluster coordination). With all nodes running at full utilization, the cluster will consume ~260 to 280 W, depending on the number of disk drives installed.

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, used in the brawny server, consumes ~10 W per DIMM.

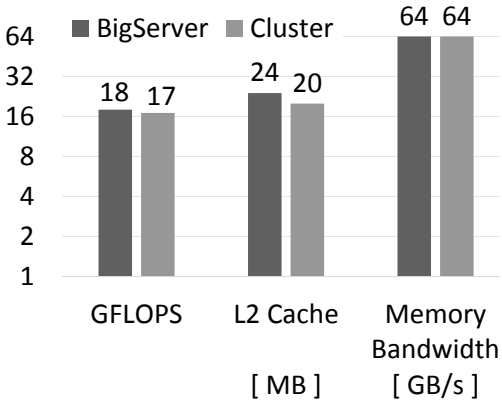


Figure 10.3: Theoretical performance figures for both systems

The power consumption of the big server (with 10 SSDs) ranges from ~200 watts when idle to ~430 watts at full utilization.[1] In theory, the systems should show similar performance. All nodes in the cluster come with 16.6 (10x1.66) GFLOPS, whereas the performance of the big server is rated with 17.6 GFLOPS. Furthermore, L2 caches and memory bandwidth of both systems are similar and the same number of disks is installed. Figure 10.2 visualizes power consumption for both systems. Of course, the relationships shown for the power consumption are idealized, because power consumption of the big server and the full cluster may differ to a certain amount. Figure 10.2(a) shows how energy proportionality drawn for the big server is much better approximated by the cluster where at least 1 and—depending on the workload—up to 10 nodes are active. Figure 10.2(b) gives an impression about the power consumption of both systems when running at a specific activity level.

In Figure 10.3, theoretical performance of big server and cluster, as given in the product sheets, is compared. CPU power, cache size and bandwidth are almost identical for both systems.

[1] These measurements include only 24 GB of DRAM as previously explained.

10.3 Experiments

To compare the energy consumption of our cluster to that of a traditional DB server, we have processed OLAP and OLTP workloads on both platforms. We have run performance-centric benchmarks first, to assess peak performance of both systems. Next, we have evaluated energy-centric benchmarks to identify the energy-efficiency potential of the big server and the cluster. In the following, we first describe the experimental setup, before we present our results.

10.3.1 Experimental setup

For all experiments, using OLTP and OLAP, we have set up the systems as previously described. A separate server, directly connected to the master node and the big server, respectively, is used as the benchmark driver, submitting queries to the cluster as well as monitoring response time and throughput. The previously introduced power measurement device is also hooked up to the benchmark driver to correlate all measurements with energy consumption.

OLAP workloads For measuring OLAP performance and energy efficiency, we are using the well-known TPC-H benchmark with a scale factor of 300; hence, 300 GB of records are generated. Due to additional indexes and storage overhead, the final DB has approx. 460 GByte of raw data. On the centralized server, small tables are stored on a single disk, whereas larger ones, e. g., the *LINEITEM* and *ORDERS* tables, are partitioned and distributed among all disks to increase access bandwidth and to parallelize processing on partitions.

On the cluster, the *REGION* and *NATION* tables are replicated to all nodes, while the other tables are partitioned and distributed equally among the nodes. In static benchmarks, no repartitioning occurs, even if the initial distribution leads to hotspots in the data, that impact the node's performance. If dynamic features of WattDB are enabled, the DBMS will automatically repartition as previously described.

OLTP workloads For online transaction processing, we are running the TPC-C benchmark on the systems with a scale factor of 1000. Hence, a thousand warehouses were generated on the cluster, consisting of about

100 GB of data. Due to indexes and storage overhead, the final DB's size is ~200 GB in the beginning of the experiments.

10.3.2 Performance-centric benchmark

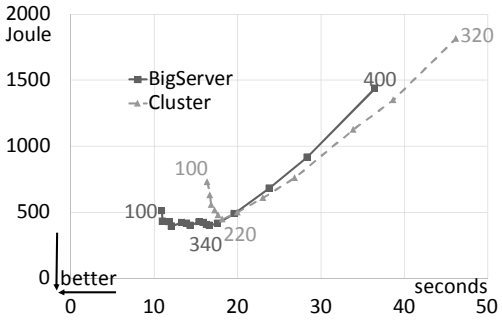
First, to evaluate the *peak performance* of both systems, we run performance-centric benchmarks similar to TPC-C and TPC-H on the cluster and the big server. We repeated the experiments with a varying number of parallel DB clients in order to estimate a saturation point, i. e., how many parallel queries the systems can process—without entering an overload state. In Figure 10.4(a), the OLAP benchmark results are depicted. The X-axis shows response time in seconds, while the Y-axis illustrates energy consumption per query. The numbers on the individual graphs annotate the number of parallel DB clients for this curve progression.

Compared to the cluster, we can conclude that the big server handles queries for the same number of clients generally faster than the cluster, it also exhibits better energy efficiency. Up to 340 DB clients, the response times on the big server increase only slightly. Then, the server seems saturated and runtimes start to build up. Consequently, energy consumption per query rises.

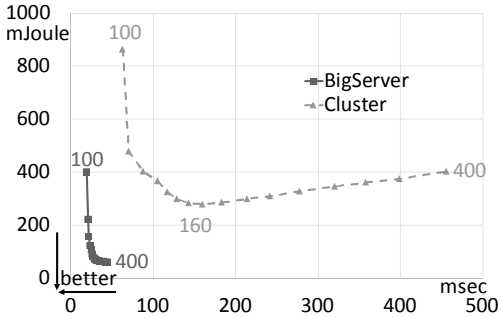
Even for medium-sized workloads (up to 220 clients), the cluster cannot meet the energy and performance figures of the big server. Already 220 clients seem to overload the cluster, where runtimes and energy consumption for the same number of clients start to increase faster than for the big server. When stressing the cluster with more than 340 clients, WattDB crashes due to shortage of main memory.

Figure 10.4(b) illustrates the results of the same experiments repeated using OLTP queries. The results reveal that the big server is much better suited for OLTP than the cluster, as is exhibits lower query response times and also less energy consumption. Query response times on the brawny server increase only slightly with the number of DB clients and the system does not show saturation at all. Consequently, energy consumption per query improves continuously. In contrast, the cluster is saturated with 160 clients; when further increasing the number of parallel queries, response times start to increase faster.

Analyzing the access patterns of both, OLTP and OLAP queries, the different performance figures are explainable: OLAP queries read



(a) OLAP: Energy/query over response time



(b) OLTP: Energy/query over response time

Figure 10.4: Peak performance and energy consumption for both systems

huge amounts of records, join them with (small) fact tables, and then group and aggregate the results to satisfy analytical inquiries. Hence, the reading part of these queries can run in parallel on all partitions, speeding up the query linearly with the number of disks, CPUs, and/or nodes. After having fetched the qualified records, the joins with the fact tables can also run concurrently. The final grouping and aggregation steps can be pre-processed locally for each of the parallel streams and quickly aggregated into a final result. Hence, this kind of access pattern seems to well fit both, a single, multi-core machine with lots of disks and memory but also a cluster of independent nodes, exchanging query

results via network. In Section 9, we have already analyzed the abilities of a cluster—with an emphasis on dynamic repartitioning of DB data—to process that kind of workloads in greater detail.

In contrast, OLTP queries touch very little data, but update records frequently. Since writers need to synchronize to avoid inconsistencies, lock information must be shared among all nodes involved. A single server with a main-memory-resident lock table can synchronize transactions much faster than a cluster, needing to exchange lock tables among nodes. Further, OLTP query operators modifying records cannot be offloaded to other nodes. Therefore, a query plan for transactional workloads is much more rigid than of OLAP queries.

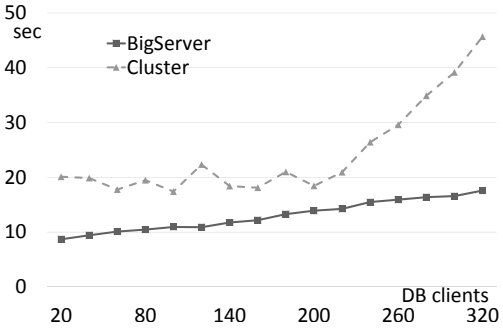
In summary, it is comprehensible that a cluster of nodes is better suited for OLAP kinds of workloads than for typical transaction processing with ACID guarantees.

10.3.3 Energy-centric benchmark

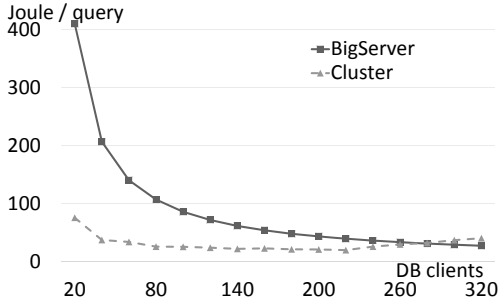
After evaluating the peak performance of both configurations, we ran experiments representing average, real-world workloads. Because DB servers are often heavily underutilized, as mentioned earlier, we modified the benchmark driver to submit queries *at timed intervals* [SHK12].

Workload scaling In each experiment, we have spawned the number of OLTP or OLAP clients between 20 and 320, sending queries to the database. Each client sends a query in a specified interval. If the query is answered within the interval, the next query is not initiated immediately, but at the start of the subsequent interval. If the query is not finished within the interval, the client waits for the answer until sending the next query. In this way, each DB client generates its share of utilization. The DBMS has to answer queries quickly enough to satisfy the DB clients, but there is no reward for even faster query evaluation. It is important to delay query submission of the clients, since we are not interested in maximizing throughput, but want to adjust the DBMS to a given workload instead, using an optimal number of nodes.[2]

[2] Otherwise, the whole benchmark would degenerate to a simple performance-centric evaluation, which is not what we intended.



(a) Query response time



(b) Energy consumption per query

Figure 10.5: Performance and energy consumption for varying OLAP utilizations

Before each workload changes, the cluster is manually reconfigured to best match the expected workload. We let the benchmarks run for a short warm-up time prior to measuring energy efficiency and performance to eliminate start-up cost and to identify maximum energy savings potential.

OLAP In this experiment, we vary the number of parallel clients between 20 and 320. As before, we are using TPC-H queries on a SF300 dataset. To control utilization, the clients send queries at an interval of

at least 20 seconds. Whatever comes last, the query result or the end of the interval, is the trigger for the next query.

Figure 10.5 illustrates the results for the energy-centric OLAP benchmark. The left side depicts query response times of the brawny server and the wimpy cluster. As expected, the centralized machine handles queries faster than the cluster, even faster than the (pre-specified) target response time of 20 seconds per query. Therefore, the server is idle for longer time periods, still consuming energy.

The cluster is meeting the target response times quite well, except for higher utilization, as observed earlier. After about 220 parallel clients, query performance starts to drop and runtimes build up. Comparing energy consumption per query of both systems, the cluster delivers far better results for *average utilization*. Due to the cluster's scale-out and adaptation to the necessary number of nodes, its energy consumption per query stays at the same level almost the entire time, regardless of utilization. Only at high workloads, energy consumption increases because of lengthy query runtimes.

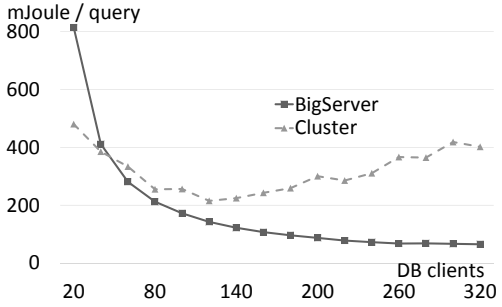
The big server, with more or less static power consumption over the whole utilization spectrum, delivers bad energy efficiency for low and moderate workloads. Only at high utilization, when all the processing power of the server is needed, its energy consumption per query pushes below that of the cluster. From this experiment, we conclude that the cluster seems to be better fit for moderate OLAP workloads than the big server.

OLTP We repeated the same experiment using OLTP queries from the TPC-C benchmark. Prior, the corresponding dataset was generated with a scale factor of 1,000. Identical to the OLAP benchmark, we scaled the DB clients between 20 and 320. Each client was waiting 40 ms between queries to simulate low and moderate workloads too.

Figure 10.6 plots the results of the energy-centric OLTP benchmark run. Whereas the big server exhibits query response times between 30 and 50 milliseconds, the cluster performs with processing times between 50 and 450 ms much worse. Apparently, the cluster is not well suited to process update-intensive OLTP workloads.

Likewise, energy efficiency of the cluster is only better at very low workloads, below 50 parallel DB clients. While the big server consumes

(a) Query response time



(b) Energy consumption per query

Figure 10.6: Performance and energy consumption for varying OLTP utilizations

between ~100 and ~800 mJoule/query, the cluster needs between ~200 and ~400 mJoule/query. The big server's energy efficiency is much better at more challenging workloads, unlike OLAP workloads, where the cluster was more efficient for most scenarios. In conclusion, we have constituted a tradeoff between performance and energy consumption on the cluster. By reducing the number of nodes, both power consumption and peak performance are lowered. For moderate workloads, lower performance is tolerable and, thus, energy efficiency can be improved.

10.3.4 Dynamic workloads

As previously described, the limiting factor for dynamic repartitioning is the migration cost, i. e., the performance impact and time it takes to move data between nodes. To estimate its impact on the cluster's elasticity, we have run experiments on a *dynamically adapting* cluster. Similar to the previous tests, we are running a mix of workloads against the cluster, ranging from low utilization up to heavy workloads. In this experiment, the cluster is given no warm-up times to adjust itself to a given task; instead, we are monitoring performance and energy consumption continuously.

Workloads change every 5 minutes, starting with a moderate workload of 20 DB clients, sending OLTP or OLAP queries respectively. The workload pattern is depicted underneath all result figures.

To quantify the importance of forecastable workloads, we have run the same workloads on the cluster twice—and once on the big server for comparison. The first run on the cluster hits the system unprepared; WattDB has to reactively adjust to the changing workload. After that, the same benchmark is run again, this time informing the DBMS of upcoming workloads (30 minutes in advance). Hence, the DBMS may use this information to proactively adjust to such a situation.

In [KHH12], we have developed a load forecasting framework to predict upcoming database workloads based on historic performance data. We observed that workloads are often repetitive and, therefore, quite easy to forecast.

OLAP and OLTP processing

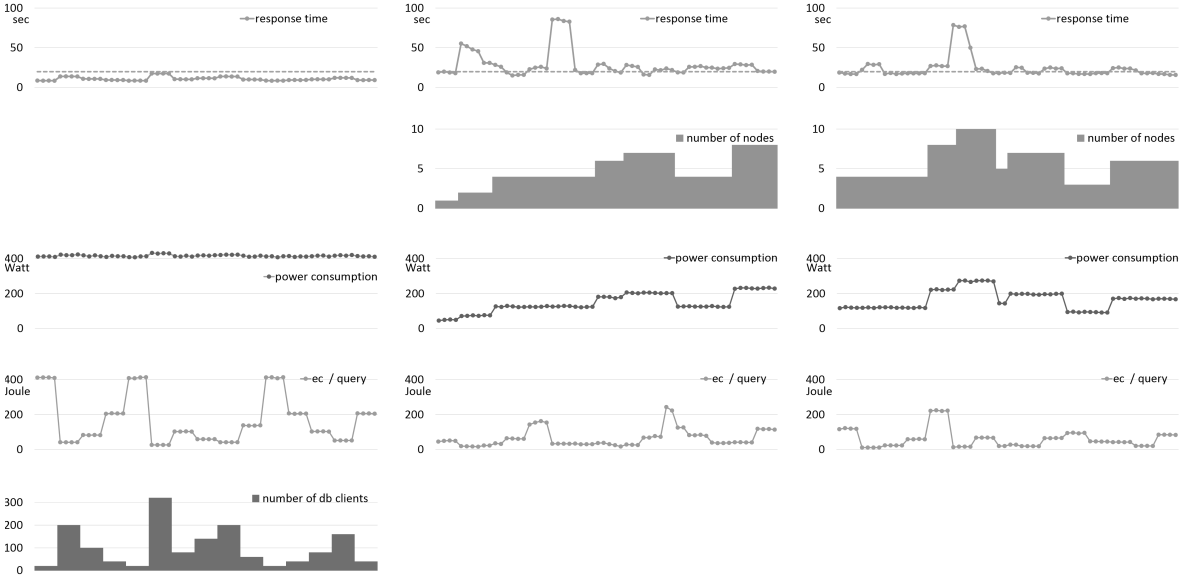
In the following, results of the benchmark runs are discussed separately for OLAP and OLTP. Results for the big server are depicted on the left-hand side of figures 10.8 and 10.7. In the middle part, the plots represent benchmark results measured on the cluster without workload forecasting. The right-hand side illustrates results of the same experiment using forecasting. The top-most plot in every column draws the average query response time. The target response time of 20 seconds is included to expose the load-dependent response time deviations in the various experiments. To characterize the varying size of the cluster, the number of active nodes is visualized. Underneath, the course of the

overall power consumption is shown for all three experiments. The resulting average energy consumption per query is plotted in the graphs below—to contrast it to the power consumption of the corresponding system. The last charts in each column visualizes the workload mix (which was the same for all three experiments).

OLAP (big server) Figure 10.7, leftmost column, shows the results for TPC-H queries on the big server. The big server does not exhibit transition times between workload changes, since reconfiguration is not needed on this single-node system. Query runtimes are fast, always beating the target response time. Yet power consumption is constantly high, regardless of utilization, as already observed in earlier experiments. Average energy consumption per query is comparably high, although query runtimes are low. Because this benchmark is energy-centric, faster query runtimes do not lead to better results.

OLAP (cluster) The middle column of Figure 10.7, depicts the TPC-H results for the cluster without workload forecasting. The number of nodes in the cluster jitters heavily, as the system tries to adjust itself to the current workload. Reconfiguration takes time, e. g., migrating from 2 to 4 nodes requires each of the two source nodes to ship about 100 GB of data to one of the targets, hence, it takes up to 20 minutes to complete repartitioning. Therefore, query response times in this benchmark experiment are highly fluctuating and, in turn, often missing the predefined deadline (target response time). Yet, as we have shown in previous experiments, the cluster, in theory, should be able to handle most of the workloads within the deadline. Due to high additional reconfiguration overhead, the nodes are overloaded. Therefore, query runtimes and average energy consumption per query remain high.

OLAP (cluster with forecasting) In this benchmark, we inform the cluster of workload changes occurring in the next 30 minutes. Hence, instead of only reacting to load changes, WattDB can now prepare for upcoming load. The plots on the rightmost column of Figure 10.7 illustrate the results. In comparison to the first run on the cluster, response times are generally lower and pass the deadline more often. Because the cluster prepares for heavy workloads in advance by scaling out to



(m) Results on the big server (n) Results without forecasting (o) Results using forecasting

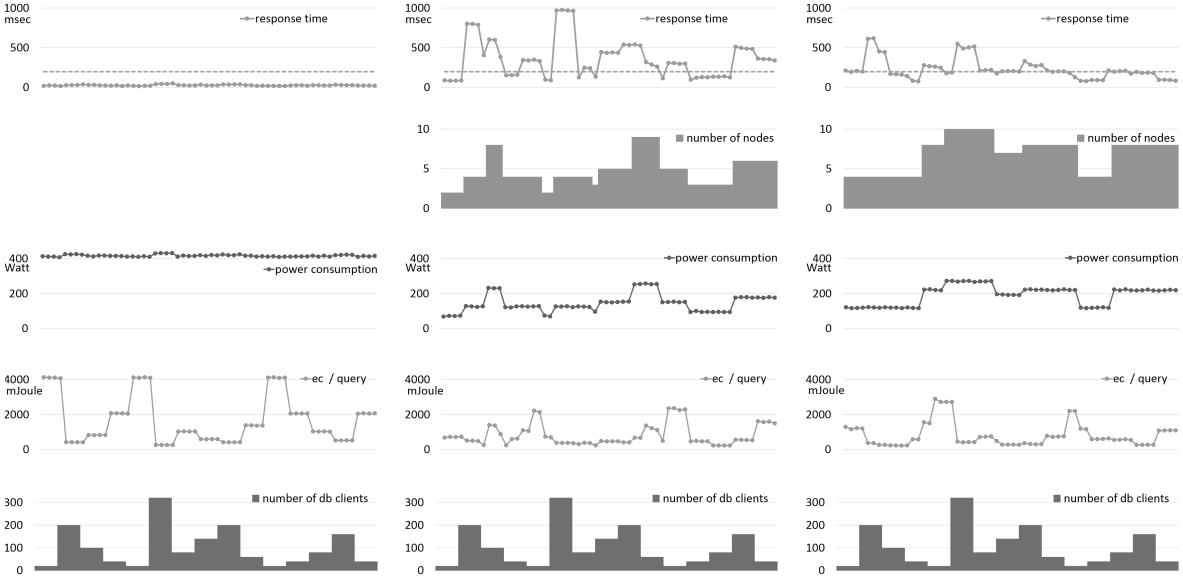
Figure 10.7: Dynamic OLAP workload on the big server and the cluster

more nodes, the number of nodes is also larger in average, resulting in increased average power consumption. The energy consumption per query, however, shows a mixed picture. For low utilizations, but more nodes running to prepare for upcoming events, energy consumption is higher compared to the version without forecasting. On the other hand, for higher utilization, thanks to in-advance preparations related to the forecasting approach, query runtimes are lower and exhibit better overall energy efficiency.

When comparing the big server with the cluster, we can conclude that the server is performing better, i. e., it exhibits lower query response times. On the other hand, the cluster is more energy efficient, especially during low and moderate utilization, due to its adaptation to the workload. The cluster benefits from scale-in, when performance is not needed. This translates to a steadily varying power consumption (according to the cluster size), whereas the server displays a more or less constant one. For OLAP workloads, the cluster seems like an eligible alternative to a big server.

OLTP (big server) After running OLAP benchmarks, we have repeated the same dynamic workload with OLTP clients on the TPC-C dataset. Figure 10.8 illustrates our results for experiments based on the energy-centric OLTP benchmark. As before, the left column characterizes the behavior of the big server, which exhibits low query runtimes, but also high power consumption.

OLTP (cluster) The middle column of Figure 10.8 summarizes our results for the benchmark run on a non-forecasting cluster. Obviously, the response times shown are high. Because the cluster is forced to permanently repartition, response times and, in turn, energy efficiency are further worsened. Because the cluster can only react to the current workload, rebalancing starts after a sufficient change of the workload is detected. As discussed for the OLAP benchmark, this puts too much stress on the nodes and notably slows down query processing. Compared to the big server, query response times are much longer for high activity levels of the cluster. Yet, power and energy consumption are lower. Therefore, the cluster delivers better energy efficiency overall—if longer query response times are deemed acceptable.



(m) Results on the big server (n) Results without forecasting (o) Results using forecasting

Figure 10.8: Dynamic OLTP workload on the big server and the cluster

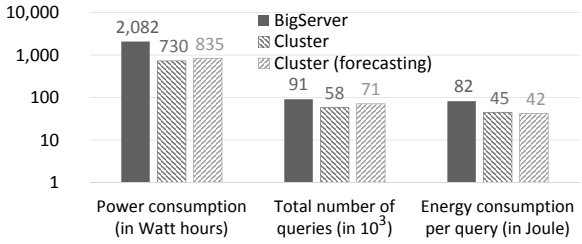
OLTP (cluster with forecasting) The right-most column of Figure 10.8 plots the OLTP benchmark results on the cluster using forecasting. Compared to the previous benchmark, the average number of nodes is higher, because WattDB is preparing for workloads in advance. As a result, query runtimes are more stable and more often pass the deadline. However, power consumption is often higher. Again, overall energy efficiency is characterized by a mixed picture: Due to preparations related to the forcasting approach, lower workloads have worse energy efficiency, but more intense workloads benefit by achieving lower energy consumption per query.

Summary Reviewing the results from all benchmarks, we want to extract some condensed numbers to facilitate high-level comparison and to gain a few key observations. For this reason, we have separately computed indicative numbers for the dynamic OLAP and OLTP experiments: Total energy consumed (in watt hours), overall query throughput in units of 10^3 resp. 10^6, and average energy consumption in joules resp. millijoules per query. These condensed numbers are visualized in Figure 10.9, where the logarithmic Y-axis should be regarded.

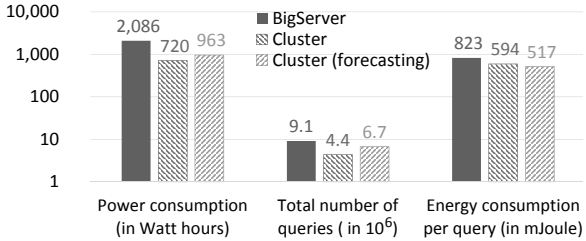
First, the cluster is no match for the big server considering pure performance. The centralized system does not require network communication or synchronization among nodes. Therefore, it can deliver much better query throughput than the cluster, where queries have to be distributed, results have to be collected, and the overall execution of concurrent queries on multiple nodes needs some form of synchronization to ensure ACID properties.

Second, the cluster handles low and moderate workloads quite well, although the big server is still faster. Yet, the cluster requires less than half of the server's power (left-most bars in the figures). Therefore, the cluster needs less energy per query and is more energy efficient, as depicted by the right-most bars in the figures.

Third, dynamic workloads with varying utilization require preparation to adjust the number of nodes to the needs. If workloads are predictable, the cluster exhibits better energy efficiency than the single server while delivering comparable performance. Although, energy consumption of a forecasting cluster is higher, its query performance outweighs the additional wattage.



(a) OLAP workload



(b) OLTP workload

Figure 10.9: Overall energy consumption, throughput, and average energy consumption per query

Energy delay product

Despite different hardware platforms and power consumption figures, by calculating the EDP, we can make a square comparison of the big server and the cluster. In Figure 10.10, we have summarized the EDP results from our dynamic experiments on OLTP and OLAP workloads for all three configurations. To illustrate the differences, all values are relative to the EDP of the big server. Hence, the forecasting cluster's EDP is about half of that of the big server, regardless of OLTP or OLAP workloads. For OLTP workloads, the non-forecasting cluster exhibits nearly the same EDP as the big server, obviously, forecasting really payed off here. In contrast, the non-forecasting cluster shows the lowest EDP under OLAP queries, even lower than the forecasting version. Apparently, preparing the cluster for upcoming workloads was

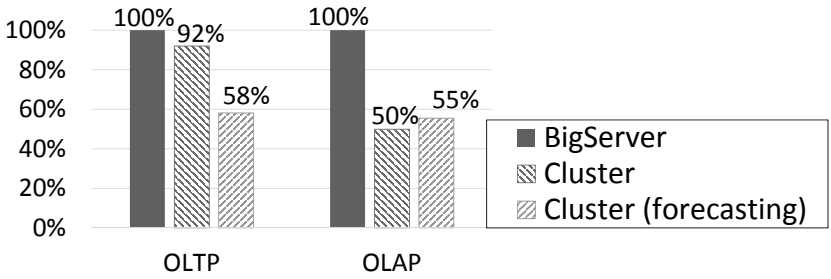


Figure 10.10: EDP of OLAP & OLTP workloads

a (little) waste of time and energy here. However, the cluster's EDP is lower than that of the big server; hence, with the same energy budget, we should be able to perform more work—although it might take some extra time.

10.4 Summary

Here, we have examined the energy-saving potential of a clustered DBMS compared to a traditional DBMS based on a single server. An important goal of this work was to compare performance and energy efficiency of our WattDB cluster to those of a big server. Of course, if peak DBMS performance is required during almost the entire operating time, a single-server approach has no alternative as our performance-centric benchmarks clearly reveal. However, as stated in various studies [BH09; SHK12], average utilization figures are far from continuous peak loads. A large share of database or data-intensive applications runs less than an hour close to peak utilization on workdays and is resilient w.r.t. somewhat slower response times. During the remaining time, their activity level is typically in the range of 20–50% and often lower. Therefore, from low- to mid-range workloads, a dynamically adjusting cluster of nodes will consume significantly less power without sacrificing too much performance. Hence, their throughput/response time requirements could be conveniently satisfied by the performance characteristics of our cluster with much less energy consumption, as confirmed by Figure 10.7. Hence, the application range, where the

cluster's energy efficiency largely dominates that of a single server, has quite some practical benefit.

Especially for OLAP workloads, where lots of records need to be read and aggregated without much coordination effort, a cluster seems to be a viable alternative to a single server. On the other hand, when processing OLTP workloads, where transactions need to synchronize continuously, a cluster suffers from significant friction losses and handles heavy workloads a magnitude slower than the centralized approach.

As shown, predictability of workloads and data elasticity are crucial for our approach. Fortunately, typical usage patterns are predictable and a cluster can therefore prepare for upcoming workloads. Thus, dynamically adjusting a cluster to the workload—although time-consuming—is possible.

11 Summary and Future Work

On the final pages of this dissertation, it is time to look back upon the recent years of research and the outcome, represented and summarized in this work. Here, we summarize our findings and give an outlook on related questions, that we had to leave open.

11.1 Conclusion

Energy efficiency is an emerging concern in all areas of IT. Starting with mobile, battery-driven equipment like mobile phones and autonomous sensors, approaches to reduce power and energy consumption has arrived in datacenters.

In this thesis, we have summarized our findings over the past years of research on energy efficiency in databases. We started our exploration by developing a measurement framework to quantify power consumption, developed new benchmarking paradigms reflecting more realistic usage patterns, and finally started to work on our own implementation of an energy-proportional DBMS.

First, we examined 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 is dependent on 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 scaling in as needed, our vision of an energy-proportional DBMS has come true.

In this work, we have proven the possibility to save energy with a distributed, elastic DBMS running on dedicated, lightweight nodes. Yet,

we had to leave many stones unturned for the sake of developing a proof-of-concept prototype. In the end, our goal to develop an energy-proportional DBMS can be deemed a success.

11.2 Outlook

As already indicated, is it not possible to completely cover all aspects of energy efficiency in a single thesis. Many questions worth examining are left untouched, due to time and budget constraints. In the following, some unresolved topics open for future research are presented.

Complex queries and transactions

Our DBMS supports basic query operators, various aggregation, and arbitrary join operations. These features are sufficient to run OLTP and OLAP experiments very similar to TPC-C and TPC-H.

To support more advanced queries, e. g., *EXISTS* predicates, or other subqueries, and partial joins, appropriate query operators need to be implemented. Integrating more operators should be straightforward and not require much effort.

At the transaction level, WattDB currently supports single-query transactions. Again, this choice was merely to reduce implementation cost and can be easily extended to full transactional support.

Advanced schema support

WattDB supports simple schema definitions, e. g., column types and foreign-key references. Yet, the latter are only indicators for the query optimizer, how tables relate to each other, and are not checked while executing transactions.

Implementing more advanced features like fully-enforced foreign-key restrictions or assertions would require more implementational overhead.

Again, we argue that our goal was not to design and develop a robust, fully-features DBMS.

Data replication

In our research, we have omitted data replication aspects at all. For one reason, because replicating data increases power consumption and storage cost, which contradicted our goal of reducing energy consumption. Second, replication increases fault tolerance, an aspect we were able to ignore in a small-scale cluster.

Yet, sophisticated replication might increase the system's performance by parallelizing queries and, thus, reduce energy consumption—an aspect we did not explore. Here, we are referring to other publications that examined similar problems, e. g., [Lam+02], [HHS05], and of course [ÖV11].

Scalability

The size of our experimental cluster, consisting of 10 nodes at maximum, can be considered rather small, especially when comparing our system to potentially thousands of nodes in a Map/Reduce cluster [Yan+07]. Scaling our cluster to a larger number of nodes has serious implications on hardware failure rates, therefore, replication and other means of fault tolerance are needed. Also, in our implementation, all nodes were communicating in a single broadcasting domain, connected to the same switch. Network bottlenecks were never an issue in our experiments, but an increasing number of nodes needs more sophisticated access and partitioning patterns to mitigate the limited bandwidth of a single network switch. A hierarchical approach seems promising, where query evaluation and data placement algorithms have to take the network layout into account. Yet, this question is open for further investigation.

Cloud computing

During our research, interesting, parallel development in the community started out, as colleagues began porting database functionality into the cloud. While our work focused on a distributed DBMS running on dedicated, physically present machines, we were influenced by techniques stemming from the cloud community. Likewise, our implementation could run atop IAAS clouds [FK03], eliminating the need to explicitly suspend and wake-up nodes. The cloud management layer could provide

theses services for WattDB, while the master node only needs to request and release machines. As pointed out, related approaches on cloud architectures exist, e. g., [Das11], that are very similar to our approach taken.

A Appendix

A.1 WattDB's Evolution

WattDB is a research prototype of a distributed, energy-efficient DBMS. As with every large software project, it has undergone many changes and evolved over time. Therefore, our initial experiments are based on an implementation of WattDB that is very different from later revisions.

Work on WattDB started with a prototype used for the SIGMOD Programming Contest 2010 [Gen+10] back in 2009. The goal was to implement a distributed query engine running read-only queries on a cluster of nodes. After participating in the contest, we gradually added new features to our prototype, giving it more and more query evaluation functionalities and introducing energy-related aspects. We also developed an energy measurement framework, capable of assessing the power and energy consumption of each node in the cluster individually.

In the following, we give an overview of various aspects and features of the system and it's surrounding eco-system, that have been implemented over time and were introduced into WattDB at later stages. The two time lines on the following pages sketch the evolution of WattDB over time. On top, the years are shown from the very first start of our work on WattDB's predecessor for the SIGMOD Programming Contest 2010 in the year 2009, to the latest publications in 2015. Below, our contributions and publications are listed in chronological order. Each publication is linked with a version of WattDB, that was used to generate the respective results.

The individual features and their approximate time of developemt is also plotted underneath. The time of implementation into WattDB is depicted as a link between the feature and the corresponding version. Some features are incremental improvements or replacements, implemented earlier and, thus, are linked to show their connection.

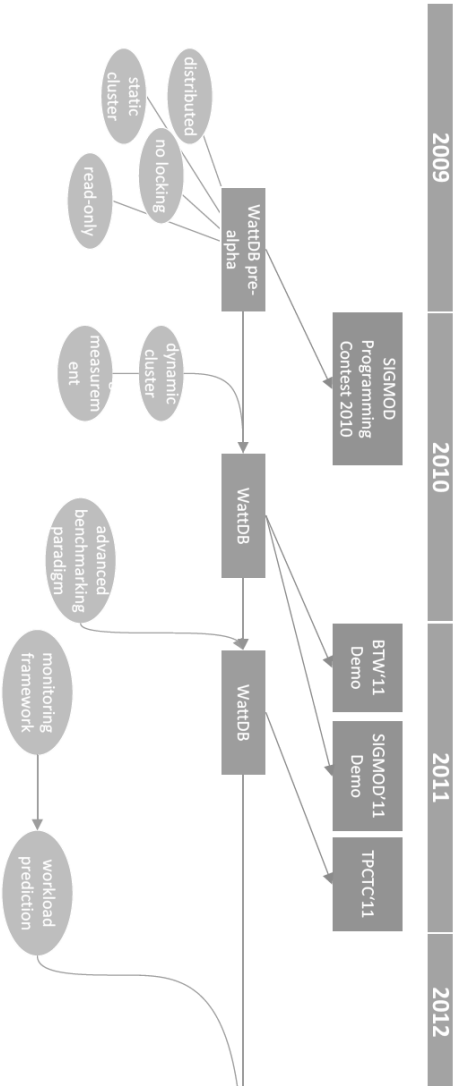


Figure A.1: Evolution of WattDB, 2009 to 2012

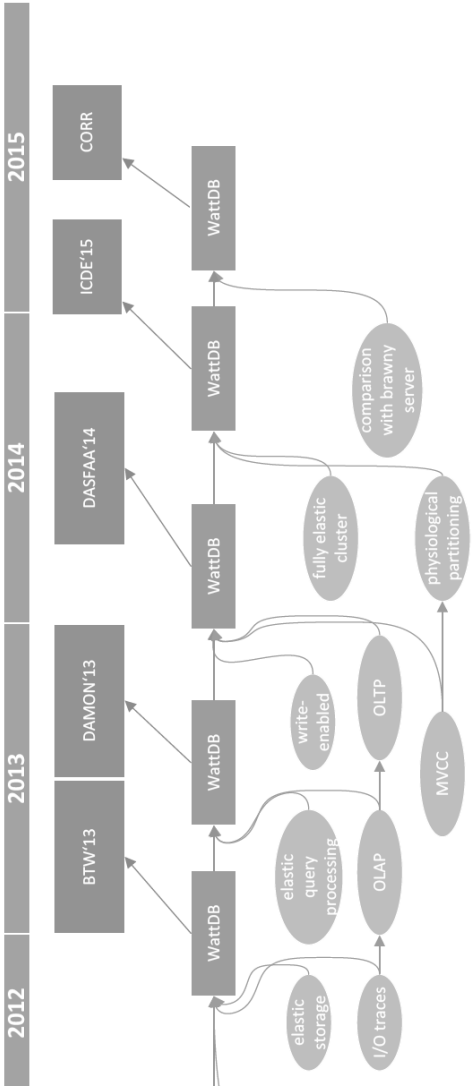


Figure A.2: Evolution of WattDB, 2012 to 2015

Listing A.1: TPC-H Q1: Pricing Summary Report Query

```
SELECT
        l_returnflag,
        l_linestatus,
        sum(l_quantity) as sum_qty,
        sum(l_extendedprice) as sum_base_price,
        sum(l_extendedprice*(1-l_discount)) as sum_disc_price,
        sum(l_extendedprice*(1-l_discount)*(1+l_tax))
                        as sum_charge,
        avg(l_quantity) as avg_qty,
        avg(l_extendedprice) as avg_price,
        avg(l_discount) as avg_disc,
        count(*) as count_order
FROM
        lineitem
WHERE
        l_shipdate <= '1998-12-01' - interval '[DELTA]' day (3)
GROUP BY
        l_returnflag, l_linestatus
ORDER BY
        l_returnflag, l_linestatus;
```

A.2 TPC-H Queries

A.2.1 TPC-H Q1

Q1, the *Pricing Summary Report Query*, is an I/O- and CPU-intensive query, scanning all data in the LINEITEM relation and selecting some records. The records are then sorted and aggregated. To leverage the high number of CPU cores in the cluster, sorting is split into two phases: First, all records are pre-sorted on the originating storage node. This done using an external sort algorithm, spilling the intermediate sort runs to a Solid-State Disk. Second, the pre-sorted records are streamed to a processing node, where all runs from the partitions are sorted using merge sort. Additionally, the now sorted records are grouped and aggregated. Listing A.1 shows the SQL query as defined in [TPC13, p. 28].

Listing A.2: TPC-H Q4: Order Priority Checking Query

```
select
        o_orderpriority, count(*) as order_count
from
        orders
where
        o_orderdate >= date '[DATE]'
        and o_orderdate < date '[DATE]' + interval '3' month
        and exists (
                select
                        *
                from
                        lineitem
                where
                        l_orderkey = o_orderkey
                        and l_commitdate < l_receiptdate)
order by        o_orderpriority
group by
        o_orderpriority;
```

A.2.2 TPC-H Q4

Query 4 is called *Order Priority Checking Query*. In the TPC-H specifications, it contains an EXISTS subquery, which is unnested by WattDB's optimizer and replaced by an equi-join, which produces identical results with superior performance. The records of the inner (ORDERS) and outer (LINEITEM) relation are accessed via an index range scan. We have chosen a hash join, because the inner relation is fitting well into main memory. Using the result of the join, the records are aggregated as defined by TPC-H. Listing A.2 plots the nested SQL query as defined in the specification [TPC13, p. 34].

A.3 Record Server

Here, additional results on different storage configurations are presented, that were not included in the main section, discussing query operator distribution.

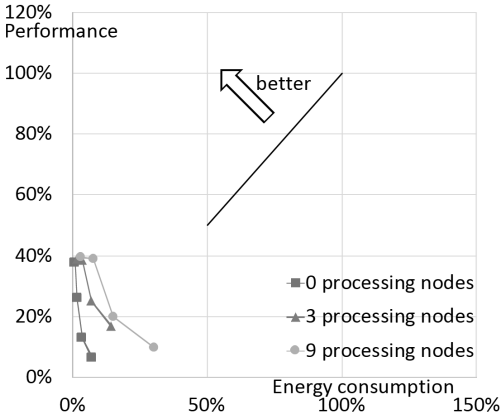


Figure A.3: TPC-H Q1 on a 1-node storage cluster

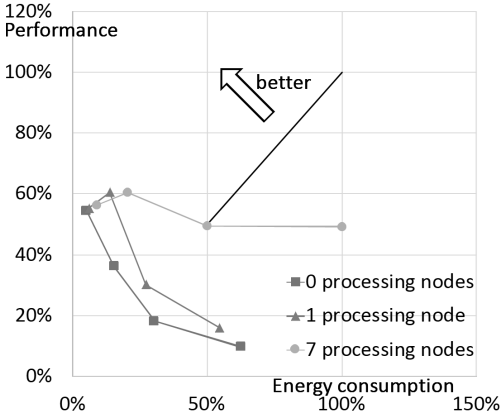


Figure A.4: TPC-H Q1 on a 3-node storage cluster

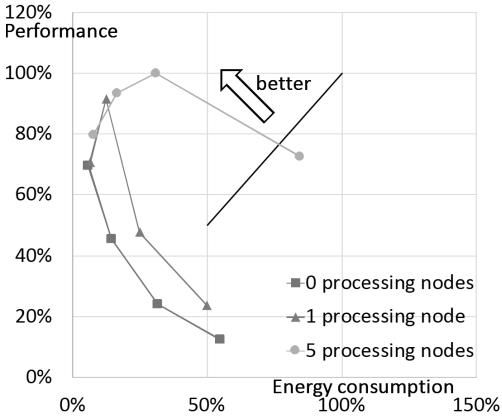


Figure A.5: TPC-H Q1 on a 5-node storage cluster

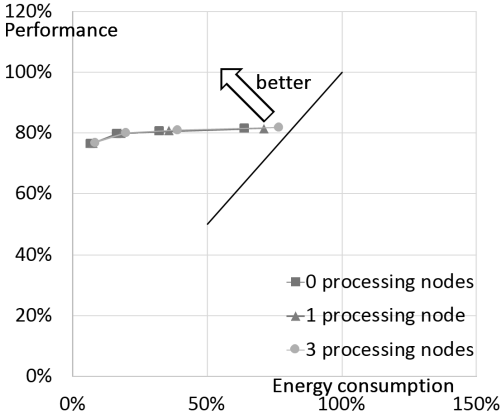


Figure A.6: TPC-H Q1 on a 7-node storage cluster

Bibliography

[ABS06] Eugene A. Avallone, Theodore Baumeister Baumeister, and Ali M. Sadegh. *Marks' Standard Handbook for Mechanical Engineers*. McGraw-Hill Professional, Dec. 2006.

[Amd13] Gene M. Amdahl. "Computer Architecture and Amdahl's Law". In: *Computer* 46.12 (Dec. 2013), pp. 38–46.

[And+09] David G. Andersen et al. "FAWN: A Fast Array of Wimpy Nodes". In: *Proceedings of the 22nd SIGOPS Symposium on Operating Systems Principles*. SOSP '09. Big Sky, MT, USA: ACM, 2009, pp. 1–14.

[Arm+09] Michael Armbrust et al. "SCADS: Scale-Independent Storage for Social Computing Applications". In: *Proceedings of the 4th Biennial Conference on Innovative Data Systems Research*. CIDR '09. Asilomar, CA, USA, 2009.

[Aye03] John E. Ayers. *Digital Integrated Circuits: Analysis and Design*. 1st Edition. CRC Press, Dec. 2003.

[Bäc13] Sebastian Bächle. "Separating Key Concerns in Query Processing - Set Orientation, Physical Data Independence, and Parallelism". PhD thesis. University of Kaiserslautern, Apr. 2013.

[Bay71] Rudolf Bayer. "Binary B-trees for Virtual Memory". In: *Proceedings of the 2nd Special Interest Group on File Description & Translation Workshop on Data Description, Access and Control*. SIGFIDET '71. San Diego, CA, USA: ACM, 1971, pp. 219–235.

[BEO10] Erwin Böhmer, Dietmar Ehrhardt, and Wolfgang Oberschelp. *Elemente der angewandten Elektronik: Kompendium für Ausbildung und Beruf*. Vieweg+Teubner Verlag, Oct. 2010.

[BG83] Philip A. Bernstein and Nathan Goodman. "Multiversion Concurrency Control—Theory and Algorithms". In: *Transactions on Database Systems* 8.4 (Dec. 1983), pp. 465–483.

[BH07] Luiz A. Barroso and Urs Hölzle. "The Case for Energy-Proportional Computing". In: *Computer* 40.12 (Dec. 2007), pp. 33–37. URL: `www.barroso.org/publications/ieee_computer07.pdf`.

[BH09] Luiz A. Barroso and Urs Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. 1st Edition. Morgan & Claypool Publishers, 2009.

[BIP06] BIPM - Bureau International des Poids et Mesures. *The International System of Units*. 8th ed. May 2006.

[BM70] Rudolf Bayer and Edward M. McCreight. "Organization and Maintenance of Large Ordered Indices". In: *Proceedings of the 1st Special Interest Group on File Description & Translation Workshop on Data Description, Access and Control*. SIGFIDET '70. Houston, TX, USA: ACM, 1970, pp. 107–141.

[BM99] Dov Bulka and David Mayhew. *Efficient C++: Performance Programming Techniques*. Addison Wesley, Nov. 1999.

[Bra+08] Matthias Brantner et al. "Building a Database on S3". In: *Proceedings of the 2008 SIGMOD International Conference on Management of Data*. SIGMOD '08. Vancouver, Canada: ACM, 2008, pp. 251–264.

[BZN05] Peter A. Boncz, Marcin Zukowski, and Niels Nes. "MonetDB/X100: Hyper-Pipelining Query Execution". In: *Proceedings of the 2nd Biennial Conference on Innovative Data Systems Research*. CIDR '05. Asilomar, CA, USA, 2005, pp. 225–237.

[Cha+06] Fay Chang et al. "Bigtable: A Distributed Storage System for Structured Data". In: *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*. OSDI '06. Seattle, WA, USA: USENIX Association, 2006, pp. 15–15.

[Coo+08] Brian F. Cooper et al. “PNUTS: Yahoo!’s Hosted Data Serving Platform”. In: *PVLDB* 1.2 (Aug. 2008), pp. 1277–1288.

[Das11] Sudipto Das. “Scalable and Elastic Transactional Data Stores for Cloud Computing Platforms”. PhD thesis. University of California, Santa Barbara, 2011.

[Dij10] Jean-Pierre Dijcks. *The Data Warehouse Insider - Partition Wise Joins*. June 2010. URL: https://blogs.oracle.com/datawarehousing/entry/partition_wise_joins.

[DIN94] DIN. *DIN 1304-1: Formelzeichen; Allgemeine Formelzeichen*. Norm. Mar. 1994.

[Dus12] Pedro M. Dusso. “A Monitoring System for WattDB: An Energy-Proportional Database Cluster”. Bachelor Thesis. University of Kaiserslautern, July 2012.

[EIA13] EIA - US Energy Information Administration. *Annual Energy Outlook (AEO) 2013*. 2013. URL: http://www.eia.gov/forecasts/aeo/pdf/0383(2013).pdf.

[EIA14] EIA - US Energy Information Administration. *Annual Energy Outlook (AEO) 2014*. 2014. URL: http://www.eia.gov/forecasts/aeo/pdf/0383(2014).pdf.

[Ele03] Electronic Specialties. *ESI #695 80A Current Probe User Manual*. 2003.

[Elm86] Ahmed K. Elmagarmid. “A Survey of Distributed Deadlock Detection Algorithms”. In: *SIGMOD Record* 15.3 (Sept. 1986), pp. 37–45.

[EPA07] EPA - US Environmental Protection Agency. *Report to Congress on Server and Data Center Energy Efficiency*. Aug. 2007. URL: http://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congress_Final1.pdf.

[Eur14] European Commission, Eurostat. *Electricity and Natural Gas Price Statistics*. May 2014. URL: http://epp.eurostat.ec.europa.eu/statistics_explained/index.php/Electricity_and_natural_gas_price_statistics.

[Fis12] Werner Fischer. “RAM Revealed”. In: *Admin Network & Security Magazine* (2012). URL: http://www.admin-magazine.com/Articles/RAM-Revealed.

[FK03] Ian T. Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Elsevier Series in Grid Computing. Elsevier Ltd., Dec. 2003.

[Flu94] Fluke Corporation. *Fluke 702 Documenting Process Calibrator Users Manual*. 1994. URL: http://assets.fluke.com/.

[Fro16] Robert Frost. “The Road Not Taken”. In: *Mountain Interval*. Henry Holt and Company, 1916.

[Gat+09] Alan F. Gates et al. “Building a HighLevel Dataflow System on top of MapReduce: The Pig Experience”. In: *PVLDB* 2.2 (2009), pp. 1414–1425.

[Gen+10] Clement Genzmer et al. “The SIGMOD 2010 Programming Contest - A Distributed Query Engine”. In: *SIGMOD Record* 39.2 (2010), pp. 61–64.

[GH96] Ricardo Gonzalez and Mark Horowitz. “Energy Dissipation in General Purpose Microprocessors”. In: *Journal of Solid-State Circuits* 31.9 (Sept. 1996), pp. 1277–1284.

[Gil02] Jeff Gilchrist. *Archive Comparison Test*. May 2002. URL: http://compression.ca/act/index.html.

[Gra+76] Jim Gray et al. “Granularity of Locks and Degrees of Consistency in a Shared Data Base”. In: *IFIP Working Conference on Modelling in Data Base Management Systems*. Freudenstadt, Germany: Elsevier, 1976, pp. 365–394.

[Gra94] Goetz Graefe. “Volcano—An Extensible and Parallel Query Evaluation System”. In: *Transactions on Knowledge and Data Engineering* 6.1 (Feb. 1994), pp. 120–135.

[Gui77] Alan E. Guile. *Electrical Power Systems*. 2nd Edition. Vol. 1. Pergamon Press, July 1977.

[Hal79] Edwin H. Hall. “On a New Action of the Magnet on Electric Currents”. In: *American Journal of Mathematics* 2.3 (Sept. 1879), pp. 287–292.

[Här+11] Theo Härder et al. “Energy Efficiency is not Enough, Energy Proportionality is Needed!” In: *Proceedings of the 1st International Workshop on Flash-based Database Systems, DASFAA Workshops*. Vol. LNCS 6637. FlashDB ’11. Hong Kong, China, 2011, pp. 226–239.

[Här+95] Theo Härder et al. “Workstation/Server-Architekturen für datenbankbasierte Ingenieuranwendungen (in German)”. In: *Informatik Forschung und Entwicklung* 10.2 (1995), pp. 55–72.

[Här05] Theo Härder. “DBMS Architecture - The Layer Model and its Evolution”. In: *Datenbank-Spektrum* 13 (May 2005), pp. 45–57.

[HD90] Hui-I Hsiao and David J. DeWitt. “Chained Declustering: A New Availability Strategy for Multiprocessor Database Machines”. In: *Proceedings of the 6th International Conference on Data Engineering*. ICDE ’90. Los Angeles, CA, USA: IEEE, 1990, pp. 456–465.

[HHS05] Hai Huang, Wanda Hung, and Kang G. Shin. “FS2: Dynamic Data Replication in Free Disk Space for Improving Disk Performance and Energy Consumption”. In: *Proceedings of the 20th Symposium on Operating Systems Principles*. SOSP ’05. Brighton, United Kingdom: ACM, 2005, pp. 263–276.

[HR83] Theo Härder and Andreas Reuter. “Principles of Transaction-oriented Database Recovery”. In: *Computing Surveys (CSUR)* 15.4 (Dec. 1983), pp. 287–317.

[HRW10] David Halliday, Robert Resnick, and Jearl Walker. *Fundamentals of Physics*. 9th Edition. John Wiley & Sons, Mar. 2010.

[HS11a] Volker Hudlet and Daniel Schall. “Measuring Energy Consumption of a Database Cluster”. In: *Proceedings of 14. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" Datenbanksysteme für Business, Technologie und Web*. BTW ’11. Kaiserslautern, Germany: GI, 2011, pp. 734–737.

[HS11b] Volker Hudlet and Daniel Schall. “SSD != SSD - An Empirical Study to Identify Common Properties and Type-specific Behavior”. In: *Proceedings of 14. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" Datenbanksysteme für Business, Technologie und Web*. BTW ’11. Kaiserslautern, Germany: GI, 2011, pp. 430–441.

[Inc12] Gartner Inc. *Gartner IT Glossary > Total Cost of Ownership (TCO)*. Dec. 2012. URL: http://www.gartner.com/it-glossary/total-cost-of-ownership-tco.

[Int04] Intel Corporation. *Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor*. Mar. 2004. URL: http://download.intel.com/design/network/papers/30117401.pdf.

[Jun+08] Hoyoung Jung et al. “LRU-WSR: Integration of LRU and Writes Sequence Reordering for Flash Memory”. In: *Transactions on Consumer Electronics* 54.3 (2008), pp. 1215–1223.

[KHH12] Christopher Kramer, Volker Höfner, and Theo Härder. “Load Forcasting for Energy-Efficient Distributed DBMSs (in German)”. In: *Proceedings of the 42nd Jahrestagung der Gesellschaft für Informatik e.V. (GI) 2012*. LNI 208 (Sept. 2012), pp. 397–411.

[Kim+08] Wonyoung Kim et al. “System Level Analysis of Fast, Per-Core DVFS Using On-Chip Switching Regulators”. In: *Proceedings of the 14th International Symposium on High Performance Computer Architecture*. HPCA ’08. Computer Society, 2008, pp. 123–134.

[Kon13] Tony Kontzer. *Data Center Operators Flock To Cold Climates*. Tech. rep. Sept. 2013. URL: http://www.networkcomputing.com/data-centers/data-center-operators-flock-to-cold-climates/d/d-id/1234459?.

[Koo+07] Jonathan Koomey et al. *A Simple Model for Determining True Total Cost of Ownership for Data Centers*. Tech. rep. Uptime Institute, 2007. URL: http://wdminc.com/whitepapers/SimpleModelDeterminingTrueTCO.pdf.

[Koo11] Jonathan G. Koomey. "Growth in Data Center Electricity Use 2005 to 2010". In: *Oakland, CA: Analytics Press. August* 1 (Aug. 2011). URL: http://www.mediafire.com/file/zzqna34282frr2f/koomeydatacenterelectuse2011finalversion.pdf.

[Kur01] Tadahiro Kuroda. "CMOS Design Challenges to Power Wall". In: *Proceedings of the 2001 International Microprocesses and Nanotechnology Conference*. MCN '01. Shimane, Japan: Business Center for Academic Societies, 2001, pp. 6–7.

[Lab08] LabJack Corporation. *Labjack UE9 User's Guide*. 2008. URL: http://labjack.com/support/ue9/users-guide.

[Lam+02] Houda Lamehamedi et al. "Data Replication Strategies in Grid Environments". In: *Proceedings of The 5th International Conference on Algorithms and Architectures for Parallel Processing*. ICA3PP '02. 2002, pp. 378–383.

[Lan+12] Willis Lang et al. "Towards Energy-efficient Database Cluster Design". In: *PVLDB* 5.11 (2012), pp. 1684–1695.

[LKP11] Willis Lang, Ramakrishnan Kandhan, and Jignesh M. Patel. "Rethinking Query Processing for Energy Efficiency: Slowing Down to Win the Race". In: *Data Engineering Bulletin* 34.1 (2011), pp. 12–23.

[Lom+09] David B. Lomet et al. "Unbundling Transaction Services in the Cloud". In: *Proceedings of the 4th Biennial Conference on Innovative Data Systems Research*. CIDR '09. Asilomar, CA, USA, 2009.

[LP09] Willis Lang and Jignesh M. Patel. "Towards Eco-friendly Database Management Systems". In: *Proceedings of the 4th Biennial Conference on Innovative Data Systems Research*. CIDR '09. Asilomar, CA, USA, 2009.

[LPN10a] Willis Lang, Jignesh M. Patel, and Jeffrey F. Naughton. "On Energy Management, Load Balancing and Replication". In: *SIGMOD Record* 38.4 (June 2010), pp. 35–42.

[LPN10b] Willis Lang, Jignesh M. Patel, and Jeffrey F. Naughton. *On Energy Management, Load Balancing and Replication. Extended Version, UW-CS-TR-1670*. Tech. rep. University of Wisconsin–Madison, 2010. URL: pages.cs.wisc.edu/~jignesh/ecodb/publ/repl.pdf.

[LPS10] Willis Lang, Jignesh M. Patel, and Srinath Shankar. "Wimpy Node Clusters: What About Non-Wimpy Workloads?" In: *Proceedings of the 6th International Workshop on Data Management on New Hardware, SIGMOD Workshops*. DaMoN '10. Indianapolis, IN, USA: ACM, 2010, pp. 47–55.

[MD97] Manish Mehta and David J. DeWitt. "Data Placement in Shared-nothing Parallel Database Systems". In: *The VLDB Journal* 6.1 (Feb. 1997), pp. 53–72.

[MGW09] David Meisner, Brian T. Gold, and Thomas F. Wenisch. "PowerNap: Eliminating Server Idle Power". In: *SIGARCH Computer Architecture News* 37.1 (Mar. 2009), pp. 205–216.

[Mic04] Micron Technology Inc. *Calculating Memory System Power for DDR2*. Apr. 2004. URL: http://www.micron.com/-/media/documents/products/technical%20note/dram/tn4704.pdf.

[Mic07] Micron Technology Inc. *Calculating Memory System Power for DDR3*. 2007. URL: http://www.micron.com/-/media/documents/products/technical%20note/dram/tn41_01ddr3_power.pdf.

[Mid89] Bette Midler. *Wind Beneath My Wings*. 1989.

[Mil13] Mark P. Mills. *The Cloud Begins With Coal - Big Data, Big Networks, Big Infrastructure and Big Power*. Tech. rep. Digital Power Group, Aug. 2013. URL: http://www.tech-pundit.com/wp-content/uploads/2013/07/Cloud_Begins_With_Coal.pdf.

[MK05] Lars Mieritz and Bill Kirwin. *Defining Gartner Total Cost of Ownership*. Tech. rep. Gartner Inc., Dec. 2005. URL: https://www.gartner.com/doc/487157/defining-gartner-total-cost-ownership.

[Nag84] John Nagle. "Congestion Control in IP/TCP Internetworks". In: *RFC* 896 (Jan. 1984).

[Nav14] Carl R. Nave. *HyperPhysics, Department of Physics and Astronomy, Georgia State University*. Jan. 2014. URL: http://hyperphysics.phy-astr.gsu.edu/hbase/hph.html.

[Neu11] Thomas Neumann. "Efficiently Compiling Efficient Query Plans for Modern Hardware". In: *PVLDB* 4.9 (June 2011), pp. 539–550.

[Ngu04] Binh Nguyen. *Linux Filesystem Hierarchy - /proc*. July 2004. URL: http://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html.

[Oet98] Tobias Oetiker. "MRTG: The Multi Router Traffic Grapher". In: *Proceedings of the 12th Conference on Systems Administration*. LISA '98. Boston, MA, USA: USENIX, 1998, pp. 141–148.

[OHJ09] Yi Ou, Theo Härder, and Peiquan Jin. "CFDC: A Flash-Aware Replacement Policy for Database Buffer Management". In: *Proceedings of the 5th International Workshop on Data Management on New Hardware, SIGMOD Workshops*. DaMoN '09. ACM, 2009, pp. 15–20.

[OHS10] Yi Ou, Theo Härder, and Daniel Schall. "Performance and Power Evaluation of Flash-Aware Buffer Algorithms". In: *Proceedings of the 21st International Conference on Database and Expert Systems Applications*. DEXA '10. Bilbao, Spain, 2010, pp. 183–197.

[Ora07] Oracle. *Oracle Database VLDB and Partitioning Guide 11g Release 1 (11.1) - Partitioning Concepts*. 2007. URL: http://docs.oracle.com/cd/B28359_01/server.111/b32024/partition.htm.

[Ora08] Oracle. *Oracle Database VLDB and Partitioning Guide—11g Release 2 (11.2), E25523-01*. 2008. URL: http://docs.oracle.com/cd/E11882_01/server.112/e25523/part_avail.htm.

[Ou12] Yi Ou. "Caching for flash-based databases and flash-based caching for databases". PhD thesis. University of Kaiserslautern, Aug. 2012.

[ÖV11] M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems*. 3rd Edition. Springer, June 2011.

[OW11] Thomas Ottmann and Peter Widmayer. *Algorithmen und Datenstrukturen*. 5th Edition. Spektrum Akademischer Verlag GmbH, Dec. 2011.

[Pan+11] Ippokratis Pandis et al. "PLP: Page Latch-free Shared-everything OLTP". In: *PVLDB* 4.10 (July 2011), pp. 610–621.

[Par+06] Seon-yeong Park et al. "CFLRU: a Replacement Algorithm for Flash Memory". In: *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*. CASES '06. Seoul, Korea, 2006, pp. 234–241.

[PD07] Larry L. Peterson and Bruce S. Davie. *Computer Networks: A Systems Approach*. 4th Edition. The Morgan Kaufmann series in networking. Morgan Kaufmann, Jan. 2007.

[PH13] David A. Patterson and John L. Hennessy. *Computer Organization and Design*. 5th Edition. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier Science, Oct. 2013.

[Pin+01] Eduardo Pinheiro et al. "Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems". In: *Proceedings of the 2nd Workshop on Compilers and Operating Systems for Low Power*. COLP '01. Barcelona, Spain: Kluwer Academic Publishers, 2001.

[PN08] Meikel Poess and Raghunath Othayoth Nambiar. "Energy Cost, The Key Challenge of Today's Data Centers: A Power Consumption Analysis of TPC-C Results". In: *PVLDB* 1.2 (2008), pp. 1229–1240.

[Rah93] Erhard Rahm. "Empirical Performance Evaluation of Concurrency and Coherency Control Protocols for Database Sharing Systems". In: *Transactions on Database Systems* 18.2 (1993), pp. 333–377.

[Rah94] Erhard Rahm. *Mehrrechner-Datenbanksysteme: Grundlagen der verteilten und parallelen Datenbankverarbeitung*. 1st Edition. Addison-Wesley, 1994.

[Reu12] Vítor U. Reus. "A GPU Operations Framework for WattDB". Bachelor Thesis. University of Kaiserslautern, July 2012.

[Rit11] Holger Ritter. *Projektdokumentations Energiemessgerät für 20 PCs - 4-1-0255-10*. Sept. 2011.

[Riv+07] Suzanne Rivoire et al. "JouleSort: A Balanced Energy-efficiency Benchmark". In: *Proceedings of the 2007 SIGMOD International Conference on Management of Data*. SIGMOD '07. Beijing, China: ACM, 2007, pp. 365–376.

[RS04] Robbert van Renesse and Fred B. Schneider. "Chain Replication for Supporting High Throughput and Availability". In: *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation*. Vol. 6. OSDI '04. San Francisco, CA, USA: USENIX Association, 2004, pp. 7–7.

[Sch09] Daniel Schall. "Energy Efficiency in Database Systems - Design of a Measurement and Analysis Framework (in German)". Diploma Thesis. University of Kaiserslautern, Sept. 2009.

[Sch11] Lars Scherer. "Total Cost of Ownership Analyse verschiedener Hardwareplattformen". Bachelor Thesis. University of Kaiserslautern, Oct. 2011.

[SH13a] Daniel Schall and Theo Härder. "Energy-Proportional Query Execution Using a Cluster of Wimpy Nodes". In: *Proceedings of the 9th International Workshop on Data Management on New Hardware, SIGMOD Workshops*. DAMON '13. New York, NY, USA: ACM, 2013, pp. 1–6.

[SH13b] Daniel Schall and Theo Härder. "Towards an Energy-Proportional Storage System using a Cluster of Wimpy Nodes". In: *Proceedings of 15. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" Datenbanksysteme für Business, Technologie und Web*. Vol. LNI 214. BTW '13. Magdeburg, Germany: GI, 2013, pp. 311–325.

[SH14a] Daniel Schall and Theo Härder. "Approximating an Energy-Proportional DBMS by a Dynamic Cluster of Nodes". In: *Proceedings of 19th International Conference on Database Systems for Advanced Applications*. Vol. LNCS 8421. DASFAA '14. Bali, Indonesia, 2014, pp. 297–311.

[SH14b] Daniel Schall and Theo Härder. "Energy and Performance—Can a Wimpy-Node Cluster Challenge a Brawny Server?" In: *Computing Research Repository* abs/1407.0386 (2014).

[SH15] Daniel Schall and Theo Härder. "Dynamic Physiological Partitioning on a Shared-nothing Database Cluster". In: *Proceedings of the 31th International Conference on Data Engineering*. Seoul, Korea: IEEE, 2015.

[Sha49] Claude E. Shannon. "Communication in the Presence of Noise". In: *Proceedings of the Institute of Radio Engineers*. Vol. 37. IRE '49. 1949, pp. 10–21.

[SHK12] Daniel Schall, Volker Höfner, and Manuel Kern. "Towards an Enhanced Benchmark Advocating Energy-Efficient Systems". In: *Proceedings of the 3rd TPC Technology Conference on Topics in Performance Evaluation, Measurement and Characterization*. Vol. LNCS 7144. TPCTC '11. Seattle, WA, USA: Springer-Verlag, 2012, pp. 31–45.

[Spe08] Alfred Z. Spector. "Distributed Computing at Multidimensional Scale". In: *Proceedings of te 9th International Middleware Conference*. Vol. LNCS 5346. MIDDLEWAR '08. Leuven, Belgium: Springer, 2008, Keynote. URL: http://middleware2008.cs.kuleuven.be/AZSMiddleware08Keynote-Shared.pdf.

[Sri+00] Jagannathan Srinivasan et al. "Oracle8I Index-Organized Table and Its Application to New Domains". In: *Proceedings of the 26th International Conference on Very Large Data Bases*. VLDB '00. Cairo, Egypt, 2000, pp. 285–296.

[SS08] Dongyoung Seo and Dongkun Shin. "Recently-evicted-first Buffer Replacement Policy for Flash Storage Devices". In: *Transactions on Consumer Electronics* 54.3 (2008), pp. 1228–1235.

[Sta13] Standard Performance Evaluation Corporation. *Server Efficiency Rating Tool (SERT) Design Document 1.0.2*. Sept. 2013. URL: http://www.spec.org/sert/docs/designdocument.pdf.

[Sta95] Standard Performance Evaluation Corporation. *SPEC Organizational Information*. 1995. URL: http://www.spec.org/spec/.

[Ste10] Angus Stevenson. *Oxford Dictionary of English*. 3rd Edition. Oxford University Press, Aug. 2010.

[Sti09] Stiftung Warentest. *Power Meters: Only one does well (in German)*. June 2009. URL: http://www.test.de/Strommessgeraete-Nur-eins-ist-gut-1781202-0/.

[Sto86] Michael Stonebraker. "The Case for Shared Nothing". In: *Database Engineering* 9.1 (1986), pp. 4–9.

[Sza+10] Alexander S. Szalay et al. "Low-power Amdahl-balanced Blades for Data Intensive Computing". In: *SIGOPS Operating Systems Review* 44.1 (Mar. 2010), pp. 71–75.

[THS10] Dimitris Tsirogiannis, Stavros Harizopoulos, and Mehul A. Shah. "Analyzing the Energy Efficiency of a Database Server". In: *Proceedings of the 2010 SIGMOD International Conference on Management of Data*. SIGMOD '10. Indianapolis, IN, USA: ACM, 2010, pp. 231–242.

[Töz+13] Pinar Tözün et al. "Scalable and Dynamically Balanced Shared-everything OLTP with Physiological Partitioning". In: *VLDB Journal* 22.2 (2013), pp. 151–175.

[TPC10a] TPC - Transaction Processing Performance Council. *TPC BENCHMARK™C Standard Specification, Revision 5.11*. Feb. 2010. URL: http://www.tpc.org/tpcc/spec/tpc-c_v5-11.pdf.

[TPC10b] TPC - Transaction Processing Performance Council. *TPC BENCHMARK™E Standard Specification, Version 1.12.0*. June 2010. URL: http://www.tpc.org/tpce/spec/v1.12.0/tpce-v1.12.0.pdf.

[TPC10c] TPC - Transaction Processing Performance Council. *TPC-Energy Specification Standard Specification, Version 1.2.0*. June 2010. URL: http://www.tpc.org/tpc_energy/spec/tpc-energy_specification_1.2.0.pdf.

[TPC13] TPC - Transaction Processing Performance Council. *TPC BENCHMARK™H Standard Specification, Revision 2.16.0*. June 2013. URL: http://www.tpc.org/tpch/spec/tpch2.17.0.pdf.

[Tu+14] Yi-Cheng Tu et al. "A System for Energy-efficient Data Management". In: *SIGMOD Record* 43.1 (May 2014), pp. 21–26.

[TWX11] Yi-Cheng Tu, Xiaorui Wang, and Zichen Xu. "Power-aware DBMS: Potential and Challenges". In: *Proceedings of the 23rd International Conference on Scientific and Statistical Database Management*. SSDBM '11. Portland, OR, USA: Springer-Verlag, 2011, pp. 598–599.

[VCO10] Hoang Tam Vo, Chun Chen, and Beng Chin Ooi. "Towards Elastic Transactional Cloud Storage with Range Query Support". In: *PVLDB* 3.1-2 (Sept. 2010), pp. 506–517.

[Ver+10] Akshat Verma et al. “SRCMap: Energy Proportional Storage Using Dynamic Consolidation”. In: *Proceedings of the 8th USENIX Conference on File and Storage Technologies*. FAST ’10. San Jose, CA, USA: USENIX Association, 2010, pp. 20–20.

[Vol08] Voltcraft. *Energy Cost Measuring Device—Energy Logger 3500 User Manual*. 2008. URL: http://www2.produktinfo.conrad.com/datenblaetter/125000-149999/125323-an-01-de-ENERGIEK_MESSGERAET_ENERGY_LOGGER_3500.pdf.

[Wan+11] Jun Wang et al. “A Survey on Energy-efficient Data Management”. In: *SIGMOD Record* 40.2 (Sept. 2011), pp. 17–23.

[WTA13] Louis Woods, Jens Teubner, and Gustavo Alonso. “Less Watts, More Performance: An Intelligent Storage Engine for Data Appliances”. In: *Proceedings of the 2013 SIGMOD International Conference on Management of Data*. SIGMOD ’13. New York, NY, USA: ACM, 2013.

[XTW10] Zichen Xu, Yi-Cheng Tu, and Xiaorui Wang. “Exploring Power-Performance Tradeoffs in Database Systems”. In: *Proceedings of the 26th International Conference on Data Engineering*. ICDE ’10. Long Beach, CA, USA, 2010, pp. 485–496.

[Yan+07] Hung-chih Yang et al. “Map-reduce-merge: Simplified Relational Data Processing on Large Clusters”. In: *Proceedings of the 2007 SIGMOD International Conference on Management of Data*. SIGMOD ’07. Beijing, China: ACM, 2007, pp. 1029–1040.

[YHJ11] Gae-won You, Seung-won Hwang, and Navendu Jain. “Scalable Load Balancing in Cluster Storage Systems”. In: *Proceedings of the 12th ACM/IFIP/USENIX International Conference on Middleware*. MIDDLEWARE ’11. Lisbon, Portugal: Springer-Verlag, 2011, pp. 101–122.

[Zhe12] Teo Zhenjie. *Power Consumption of PC Components in Watts*. 2012. URL: http://www.buildcomputers.net/power-consumption-of-pc-components.html.

Curriculum Vitae

Personal

Name	Daniel Georg Schall
Birthday	April 13, 1984
Birthplace	Worms, Germany

Education

10/2009 - 10/2014	PhD candidate Database and Information Systems Group, University of Kaiserslautern
04/2004 - 09/2009	Diploma (Dipl.-Inf.) in Applied Computer Science, University of Kaiserslautern

Work Experience

since 11/2014	Software developer SAP AG, Walldorf
10/2009 - 10/2014	Scientific staff member Database and Information Systems Group, University of Kaiserslautern
10/2008 - 04/2009	Internship IBM Software Group (Business Intelligence), Böblingen
2004 - 2008, during semester breaks	Student trainee Abbott GmbH & Co. KG, Ludwigshafen am Rhein
May 2006 - Sep 2008	Network and domain administrator Department of Ethics and Social Sciences, University of Kaiserslautern
2008 & 2009, spring semester	Teaching assistant for database lectures Database and Information Systems Group, University of Kaiserslautern