TECHNISCHE UNIVERSITÄT MÜNCHEN
Fakultät für Informatik
Lehrstuhl für Rechnertechnik und Rechnerorganisation

THE DCDB FRAMEWORK

A Scalable Approach for Measuring Energy Efficiency in HPC

AXEL AUWETER

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzende(r):	Prof. Dr. Martin Bichler
Prüfer der Dissertation:	1. Prof. Dr. Arndt Bode
	2. Prof. Dr. Dieter Kranzlmüller

Die Dissertation wurde am 23.10.2018 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 29.01.2019 angenommen.

Axel Auweter

The DCDB Framework
A Scalable Approach for Measuring Energy Efficiency in HPC

ABSTRACT

During the past decades, numeric simulation has become an increasingly popular tool for scientists and engineers from a broad variety of domains. In cases where the modeling of a problem requires large amounts of compute or memory capacity, supercomputers come into play, which provide more than thousand times the compute and memory capacity of regular desktop computers. One of the challenges when operating a supercomputer is power consumption, which accounts for a large fraction of the total cost of ownership. Since power needs to be delivered to the supercomputer and the generated heat needs to be removed from the data center, the challenge of reducing the power consumption of a supercomputer not only covers the machine itself, but also the surrounding data center. Additionally, the selection of suited algorithms and configurable hardware parameters such as the processor frequency contribute to the power efficiency of a supercomputer.

Key to reducing the overall power consumption of a supercomputer and its surrounding data center infrastructure is the continuous assessment of the system's state and the associated power consumption of individual system components: only with a detailed insight into the system's behavior, system operators and users can optimize the system's configuration and the application's algorithms for reduced power, increased performance, and thus improved energy efficiency. The system's state comprises of physical parameters, such as temperatures, voltages, fan speeds, etc. as well as operating parameters such as the application and its associated performance characteristics. Reduced costs for sensor equipment and higher temporal resolution cause more monitoring data to be generated and turn supercomputer monitoring into a data processing challenge.

This thesis introduces DCDB, the Data Center DataBase. DCDB is a framework to collect, store, and analyze time series data captured from the sensors in a supercomputer. DCDB builds on the distributed Cassandra NoSQL database. Using an efficient way of storing time series data using the Cassandra data model, DCDB installations scale almost linearly in performance with the addition of more database servers. For an efficient analysis of derived metrics, DCDB provides a feature for defining virtual sensors that combine multiple physical sensors as well as other virtual sensors using freely definable arithmetic expressions.

DCDB has been tested under real operating conditions on two prototype systems. On these systems, DCDB collected monitoring data using both, traditional system monitoring mechanisms such as IPMI and SNMP, as well as mechanisms based on custom firmware, that made use of a new push approach to collecting data at high frequency. DCDB also integrated sensor data not only from the computer system, but also from the cooling infrastructure of the data center. On one of the two systems, DCDB was configured as a distributed installation with 25 monitoring nodes. This way, DCDB could prove its scalability claims: collecting monitoring data on all nodes in parallel was performing just as fast as the collection of $1/25$ of the data on a single node.

PUBLICATIONS

The work presented in this thesis has been performed as part of the European research projects Mont-Blanc and DEEP. The two projects have received funding from the European Community's Seventh Framework Programme [FP7/2007-2013] under grant agreement numbers 288777 and 287530. Ideas and figures have appeared previously in the following project-related publications:

Auweter, A., Dózsa, G., Gelado, I., Puzovic, N., Rajovic, N. & Tafani, D. *Mont-Blanc Project Deliverable 5.3: Preliminary Report on Porting and Tuning of System Software to ARM Architecture* http://montblanc-project.eu/sites/default/files/d5.3_preliminary_report_on_porting_system_software_to_arm_architecture_v1.pdf

Dózsa, G., Auweter, A., Tafani, D., Beltran, V., Servat, H., Judit, G., Nou, R. & Radojković, P. *Mont-Blanc Project Deliverable 5.5: Intermediate Report on Porting and Tuning of System Software to ARM Architecture* http://montblanc-project.eu/sites/default/files/D5.5_Intermediate_report_on_porting...._v1.0.pdf

Tafani, D. & Auweter, A. *Mont-Blanc Project Deliverable 5.8: Prototype demonstration of energy monitoring tools on a system with multiple ARM boards* http://montblanc-project.eu/sites/default/files/MB_D5.8_Prototype%20demonstration%20of%20energy%20monitoring...v1.0.pdf

Mantovani, F., Ruiz, D., Vilarrubi Barri, O., Martorel, X., Nieto, D., Auweter, A., Tafani, D., Adeniyi-Jones, C., Gloaguen, H. & Utrera Iglesias, G. *Mont-Blanc Project Deliverable 5.11: Final report on porting and tuning of system software to ARM architecture* http://montblanc-project.eu/sites/default/files/D5.11%20Final%20report%20on%20porting%20and%20tuning.%20V1.0.pdf

Tafani, D. & Auweter, A. *Mont-Blanc Project Deliverable 7.4: Concept for energy aware system monitoring and operation* http://montblanc-project.eu/sites/default/files/D7.4_Concept_for_energy_aware_system_monitoring_andoperation_v1_reduced.pdf

Meyer, N., Solbrig, S., Wettig, T., Auweter, A. & Huber, H. *DEEP Project Deliverable 7.1: Data centre infrastructure requirements* http://www.deep-project.eu/SharedDocs/Downloads/DEEP-PROJECT/EN/Deliverables/deliverable-D7.1.pdf

Meyer, N., Wettig, T., Solbrig, S., Auweter, A., Ott, M. & Tafani, D. *DEEP Project Deliverable 7.2: Concepts for improving energy and cooling efficiency* http://www.deep-project.eu/SharedDocs/Downloads/DEEP-PROJECT/EN/Deliverables/deliverable-D7.2.pdf

Auweter, A. & Ott, M. *DEEP Project Deliverable 7.3: Preparation of energy and cooling experimentation infrastructure* http://www.deep-project.eu/SharedDocs/Downloads/DEEP-PROJECT/EN/Deliverables/deliverable-D7.3.pdf

Auweter, A. & Ott, M. *DEEP Project Deliverable 7.4: Intermediate Report on DEEP Energy Efficiency* http://www.deep-project.eu/SharedDocs/Downloads/DEEP-PROJECT/EN/Deliverables/deliverable-D7.4.pdf

Ott, M. & Auweter, A. *DEEP Project Deliverable 7.5: Final Report on DEEP Energy Efficiency* http://www.deep-project.eu/SharedDocs/Downloads/DEEP-PROJECT/EN/Deliverables/deliverable-D7.5.pdf

Additionally, ideas and figures have appeared previously in the following publications:

Auweter, A., Bode, A., Brehm, M., Huber, H. & Kranzlmüller, D. in *Information and Communication on Technology for the Fight against Global Warming* (eds Kranzlmüller, D. & Tjoa, A. M.) 18–25 (Springer, 2011)

Wilde, T., Auweter, A. & Shoukourian, H. The 4 Pillar Framework for energy efficient HPC data centers. *Computer Science-Research and Development* **29,** 241–251 (2014)

Auweter, A., Bode, A., Brehm, M., Brochard, L., Hammer, N., Huber, H., Panda, R., Thomas, F. & Wilde, T. *A case study of energy aware scheduling on SuperMUC* in *Supercomputing* (eds Kunkel, J. M., Ludwig, T. & Meuer, H. W.) (2014), 394–409

ACKNOWLEDGEMENTS

While the following thesis has been assembled solely by myself, many people have enabled and contributed to the research activities behind it. Therefore, I would like to highlight and thank a few of them in the following paragraphs.

First and foremost, I would like to express my sincerest thanks to my supervisor Prof. Dr. Dr. h.c. Arndt Bode. I have met Prof. Bode during my first year in university attending his introductory class on Computer Architecture and Computer Engineering. Since then, Prof. Bode has provided me unprecedented support, guidance, and trust by employing me, first as student tutor, later as research associate. His experience and remarkable skills in writing support letters have enabled my acceptance into several study abroad, internship, and scholarship programs. And despite being a full-time professor and head of the Leibniz Supercomputing Centre of the Bavarian Academy of Sciences and Humanities (LRZ) at the same time, he always found the time for a personal consultation.

Secondly, I would like to thank all my former colleagues and managers at LRZ as well as the staff members of Prof. Bode's chair at Technische Universität München (TUM). With no claim to completeness, I would like to thank in particular Prof. Dr. Dieter Kranzlmüller, Dr. Victor Apostolescu, and Dr. Herbert Huber for providing me with the necessary resources and guidance. Furthermore, I would like to thank Dr. Michael Ott, Dr. Torsten Wilde, Dr. Daniele Tafani, Dr. Hayk Shoukourian, Dr.-Ing. Carsten Trinitis, and Dr. Josef Weidendorfer for their valuable time during many discussions on how to make supercomputers more energy efficient.

In addition, I would like to thank my managers and co-workers at MEGWARE Computer Vertrieb und Service GmbH, namely Jürgen Gretzschel, Steffen Eckerscham, André Singer, Sebastian Siegert, and Kai Löhnig for their support during my finishing of this thesis.

Finally, I would like to thank my family for all their multi-faceted support along the way.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

AC	Alternate Current
ASIC	Application Specific Integrated Circuit
BIC	Booster Interface Card
BMC	Baseboard Management Controller
BNC	Booster Node Card
BSD	Berkeley Software Distribution
CGI	Common Gateway Interface
CMC	Chassis Management Controller
CRAC	Computer Room Air Conditioner
DC	Data Center
DCDB	Data Center DataBase
DEEP	Dynamical Exascale Entry Platform
DVFS	Dynamic Voltage and Frequency Scaling
EMB	Ethernet Mother Board
FPGA	Field-Programmable Gate Array
HPC	High-Performance Computing
HPL	High Performance Linpack
I/O	Input/Output
I2C	Inter-Integrated Circuit Bus
IC	Integrated Circuit
IEC	International Engineering Consortium
IPMI	Intelligent Platform Management Interface
ITIL	Information Technology Infrastructure Library
ITU	International Telecommunications Union
LAN	Local Area Network
LRZ	Leibniz Supercomputing Centre of the Bavarian Academy of Sciences and Humanities

MAC	Media Access Control
MSR	Machine Specific Register
OID	SNMP Object Identifier
OS	Operating System
PCIe	Peripheral Component Interconnect Express
RAPL	Running Average Power Limit
RRD	Round Robin Database
SDB	Samsung Daughter Board
SMC	System Management Controller
SNMP	Simple Network Management Protocol
SoC	System on Chip
SPI	Serial Peripheral Interface
TDP	Thermal Design Power

1 INTRODUCTION

The use of computers has revolutionized the way for scientists to conduct research today. Computers facilitate the acquisition and processing of large amounts of experimental data and facilitate the sharing and discussion of results among researchers from the same domain across the globe. In addition, researchers in academia and industry can rely on numeric simulation instead of conducting real-world experiments for the generation of new knowledge. The use of such virtual experiments is of particular interest when real experiments are too costly, too time consuming, too dangerous, or even simply impossible to conduct. Replacing actual experiments with a computer-based simulation requires that the underlying physical principles are known, proven, and modeled with sufficient accuracy. As of today, numerous scientific domains rely on numeric simulation: according to the usage statistics of the Leibniz Supercomputing Centre of the Bavarian Academy of Sciences and Humanities (LRZ), LRZ's computers support researchers from various physics branches, meteorology, computational fluid dynamics, material sciences, life sciences, chemistry, and many more [14].

High Performance Computing (HPC) or supercomputing comes into play, when the computational or memory requirements for a given numeric simulation exceed the capabilities of ordinary desktop computers. In recent years, the demand for HPC resources has been constantly increasing and according to market analysts like IDC [15] and Intersect360 [16], this trend will likely continue. In addition, the fact that the growth of supercomputer performance according to the Top500 List of Supercomputers [17] has outperformed Moore's Law [18] for many years can be seen as another indicator for the rapid growth of the HPC market.

However, with the increase in supercomputer size and performance comes an increasing demand for electrical energy. Today's supercomputers already consume power in the range of megawatts. As a consequence, increasing budgets have to be allocated for covering the electricity costs. If this trend was to continue, power costs over the lifetime of a machine could – one day – outweigh hardware investment costs. For economical and ecological reasons, it has therefore become necessary to seek for novel techniques and approaches in reducing supercomputer power consumption.

Besides the power consumption of the HPC systems themselves, the power consumption of other equipment in the data center has attracted the attention of data center operators. Due to the principle

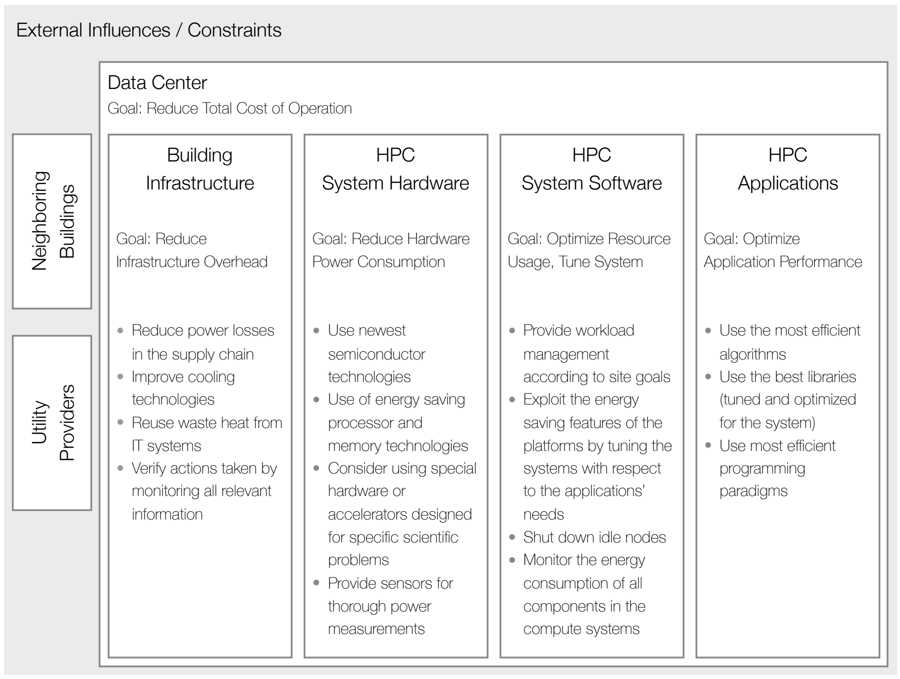

External Influences / Constraints
Neighboring Buildings
Utility Providers
Data Center
Goal: Reduce Total Cost of Operation
Building Infrastructure
Goal: Reduce Infrastructure Overhead
• Reduce power losses in the supply chain
• Improve cooling technologies
• Reuse waste heat from IT systems
• Verify actions taken by monitoring all relevant information
HPC System Hardware
Goal: Reduce Hardware Power Consumption
• Use newest semiconductor technologies
• Use of energy saving processor and memory technologies
• Consider using special hardware or accelerators designed for specific scientific problems
• Provide sensors for thorough power measurements
HPC System Software
Goal: Optimize Resource Usage, Tune System
• Provide workload management according to site goals
• Exploit the energy saving features of the platforms by tuning the systems with respect to the applications' needs
• Shut down idle nodes
• Monitor the energy consumption of all components in the compute systems
HPC Applications
Goal: Optimize Application Performance
• Use the most efficient algorithms
• Use the best libraries (tuned and optimized for the system)
• Use most efficient programming paradigms

Figure 1: 4-Pillar Framework for Energy Efficient HPC Data Centers

of energy conservation, all electrical energy supplied to a supercomputer is turned entirely into heat energy. At the expense of additional electrical energy, this excess heat has to be removed from the data center. Additionally, electrical losses in the power supply chain that incur in transformers or uninterruptible power supply units also add to the final power bill.

1.1 THE 4-PILLAR FRAMEWORK FOR ENERGY EFFICIENT HPC DATA CENTERS

To classify and structure all efforts related to energy efficiency in HPC, a framework is needed that encompasses all domains in which optimizations can be made to improve the overall energy efficiency when solving scientific problems on a supercomputer. Not many attempts for this can be found in the literature.

Beloglazov et al. [19] have created a taxonomy of energy-efficient data centers and cloud computing systems. According to their work, the contributing domains comprise of energy efficient applications, power-aware resource management systems, and energy efficient hardware. Although the data center and its infrastructure for power distribution and cooling are briefly touched in their work as part of the hardware setup, the work of Beloglazov et al. has an IT-centric approach, omitting the links between computing systems and the data center.

Valentini et al. [20] created an overview on energy efficiency techniques in cluster computing systems. However, their publication only summarizes mechanisms for static and dynamic power management and load balancing approaches. Although these have all been successfully implemented in order to reduce supercomputer power consumption, they omit the influence of the surrounding data center and the HPC applications themselves on the overall energy efficiency.

At the Leibniz Supercomputing Centre of the Bavarian Academy of Sciences and Humanities, first ideas for an all-encompassing framework for energy efficient HPC data centers were expressed in 2011 [11]. In this work, three domains for energy efficiency in HPC were highlighted: tools for energy monitoring and control, data center site infrastructure aspects, and system hardware and operation aspects. To incorporate the importance of HPC applications, this work was refined later into the 4-Pillar Framework for Energy Efficient HPC Data Centers [12] and is shown in Figure 1. The four pillars in the framework are:

1. Building Infrastructure
2. HPC System Hardware
3. HPC System Software
4. HPC Applications

At first, the 4-Pillar Framework encourages for efficiency improvements within each of these four pillars with the goal of reducing the total cost of operation. It also gives some generic recommendations such as the need for thorough system instrumentation to assist in the analysis of the system's energy efficiency in the first place.

However, for fully optimizing the energy efficiency in HPC, optimizing within each pillar might not be sufficient. For this reason and besides stressing the need for optimizations within each pillar, the 4-Pillar model also explicitly encourages cross-pillar optimizations. For example, the selection of the most efficient algorithm in a HPC application might vary depending on the type of HPC system hardware that is being used. Such cross-pillar optimizations are not only restricted to the four pillars that are within the scope of the data center itself but may even extend to external entities such as the electrical utility provider or neighboring buildings that may become users of the data center's excess heat.

The following sections give an overview of the four pillars and their role in the attempt to globally optimize the energy efficiency of high performance computing.

1.2 OPTIMIZED DATA CENTER BUILDING INFRASTRUCTURES

Data centers are functional buildings with the purpose of providing an ideal environment for IT equipment. Data center building infrastructures thus comprise of equipment to supply IT hardware reliably with electrical energy, to remove excess heat from IT equipment, to provide air-conditioning including (de-)humidification, and to ensure safe operations through physical access management and fire prevention, detection, and extinguishing systems. The extent of the measures and techniques in place in a data center varies with the priorities of the data center's goals and operators have to find a trade off between computational and application performance, reliability, safety, flexibility, and costs. Although deemed necessary for professional IT operations, data center building infrastructures consume energy on top of the IT equipment's energy usage, inevitably adding energy to the list of trade off items. Optimizing data center infrastructures therefore aims at reducing this overhead within the possibilities of the data center's operational goals.

1.2.1 *Power Distribution*

Due to their high demand, large data centers are supplied with electricity from high voltage networks (e.g. 20 kV) and operate their own transformers to reduce this to their lower working voltage (e.g. 400 V). Voltage transformation is never lossless. Still, savings can be achieved by adding or removing of parallel transformers to ensure that transformers are operating within their most efficient load range.

In addition to transformers, data centers may operate uninterruptible power supply units, which filter out voltage or phase irregularities of the power grid or supply backup power in case of a power outage. Similar to the operation of transformers, uninterruptible power supply units consume power on their own, resulting in a loss of efficiency. This can be avoided when critically reviewing the availability needs and service level agreements of each service. Particularly in high performance computing, it is often advisable to abstain from using uninterruptible power supply units. If full power outages are rare, the additional costs over a system's lifetime from restarting the supercomputing applications after a power outage are lower than the overhead costs of providing fully uninterruptible power to the supercomputer.

1.2.2 *Cooling*

When looking at the breakdown of power consumption in data center buildings, cooling typically takes the largest fraction after the IT equipment [21]. Following the second law of thermodynamics, excess

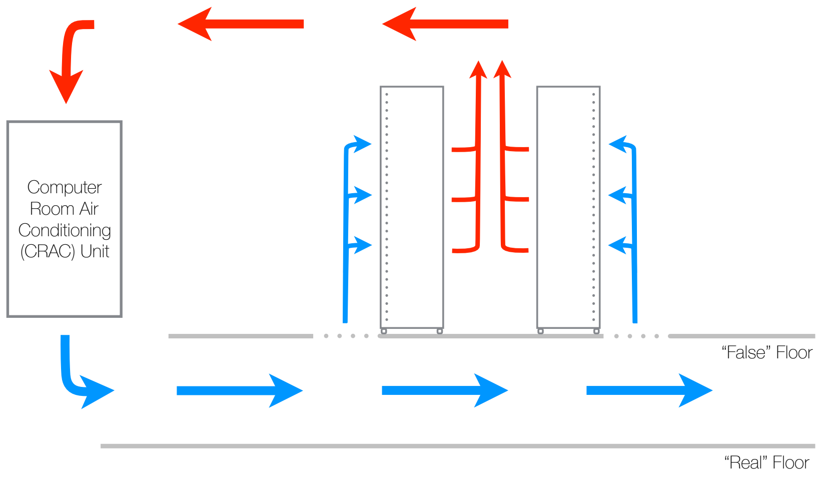


Figure 2: Common Air Cooling Scheme in Data Centers

heat must be removed from the data center by bringing in a coolant at lower temperature than the equipment to be cooled. In data centers with air cooled computing equipment, this process can be done as depicted in Figure 2. The compute equipment is located in racks standing on a "false" floor. Through openings in the false floor, cold air can exit from the area below. This cold air is then sucked in by the fans mounted in the compute equipment in order to cool the computer chips, power supplies, etc. The heated air then exits the rack on the other side and rises to the ceiling. From there, it is sucked into the Computer Room Air Conditioning (CRAC) unit which uses cold water in a water-to-air heat exchanger in order to cool the air before blowing it back into the area below the false floor.

Production of cold water to supply the CRAC units happens either through compressor based chillers, or – if outside conditions permit – directly from outside air. The latter concept is often referred to as "free cooling" although electrical energy is still required to drive pumps and cooling towers that emit the heat from the water to outside air. Thus, the term "chiller-less cooling" should be preferred.

The reason for using water as medium for heat transfer between cooling towers, chillers, and CRAC units lies in its superior thermal characteristics. A comparison of the thermal characteristics of air and water at 20°C is given in Table 1 [22]. The two major properties of interest are the *thermal conductivity* λ and the *thermal capacity* c_p. Since thermal capacity is defined as a unit relative to substance mass, it makes sense to look at the *volumetric heat capacity* which takes the density ρ of the substance into account. For the technical characterization of a cooling medium, both thermal conductivity and volumetric heat capacity play a similar role. Thus, in engineering, the definition

	Unit	*Air*	*Water*	*Factor*
Thermal Conductivity	$\frac{W}{m \cdot K}$	0.026	0.598	23 x
Thermal Capacity	$\frac{J}{g \cdot K}$	1.006	4.185	4 x
Volumetric Heat Capacity	$\frac{kJ}{m^3 \cdot K}$	1.196	4178	3493 x
Thermal Inertia	$\frac{J}{m^2 \cdot K \cdot \sqrt{s}}$	5.563	1581	284 x

Table 1: Comparison of Thermal Properties: Air vs. Water

of *thermal inertia* I has become popular, which is an artificial property defined as $I = \sqrt{\lambda \rho c_p}$.

Optimizing the cooling infrastructure in data centers can be performed in two ways. One option is to try extending the time period in a year during which chiller-less cooling is possible. This can be done either by improving the cooling tower efficiency through the use of evaporative cooling or by raising the temperature set point for the cooling water. The second option is to shorten or entirely remove air from the heat transfer chain by using direct liquid cooling technologies in which water is brought straight to the hot computer components. Also, the efficiency of compressor based chillers has been subject to constant improvements in the past. Chillers are characterized by their *energy efficiency ratio (EER)* defined as the ratio of heat energy removed E_h to the electrical energy spent E_e: $EER = {}^{E_h}/_{E_e}$. Although the EER of a chiller also depends on external factors such as the recooling temperature (which depends on the outside temperature), modern chillers should achieve an average EER of at least 5.

1.2.3 *Monitoring*

Depending on the data center size and its application-specific requirements, building infrastructures for data centers can become extremely complex. This results in an increased likelihood for misconfigurations or undetected defects that lead to reduced efficiency. The best way to cope with this complexity is a thorough monitoring infrastructure. Being able to quickly assess the status of the building infrastructure through a central infrastructure management system has become essential for data center operators.

Monitoring of electrical energy can be performed through the use of shunt resistors and Hall effect sensors. A shunt resistor is a resistor with a small resistance (R_s) compared to the resistance of the electrical load. It is placed in series with the electrical load, so that the current flow I over the shunt resistor and the load are equal. When the voltage drop U_s over R_s is measured, the current I can be derived from Ohm's law as:

$$I = \frac{U_s}{R_s} \tag{1}$$

In combination with the voltage U of the power supply, the instantaneous power P can be derived as $P = U \cdot I$ and the electrical energy E is defined as the integral of P over time: $E = \int P dt$.

Hall effect sensors make us of the Lorentz force that diverts electrons in a conductor to one side, if the conductor is surrounded by a perpendicular magnetic field. The diversion of electrons results in a measurable electric potential U_H:

$$U_H = A_H \frac{I \cdot B}{d} \tag{2}$$

with:

- A_H: material dependent hall coefficient
- I: electrical current
- B: magnetic flux density
- d: thickness of the conductor (parallel to B)

This means that for hall effect based sensors with constant A_H, B, and d values, the measurement signal U_H is proportional to the electrical current I. Thus, power and energy can be derived in similar fashion to shunt resistor based measurements.

For the instrumentation of cooling loops, heat meters can be used. Heat meters calculate the amount of thermal energy transferred from a system based on the inlet and outlet temperatures, as well as the flow rate of the heat transfer medium. The instantaneous heat power P_H in a cooling loop can then be calculated as:

$$P_H = V \cdot \rho \cdot c_w \cdot (t_{out} - t_{in}) \tag{3}$$

with:

- V: flow rate of the coolant
- ρ: density of the coolant
- c_w: specific heat capacity of the coolant
- t_{out}: outlet temperature
- t_{in}: inlet temperature

Component	*Details*	*Power*
CPU	2x Xeon® E5-2697v3	297 W
Main board	S2600WT2	64 W
Memory	8x 8GB DDR4 RDIMM	26 W
High speed interconnect	FDR InfiniBand	9 W
Others	chassis fans, power supply, ...	151 W

Table 2: Power breakdown of a Intel® R1XXX server system

The amount of thermal energy removed via a cooling loop is the integral of P_H over time: $E = \int P_H dt$.

Similar to any other monitoring system, special care has to be taken before trusting the numbers provided by any sensor. All sensors of the infrastructure management system should be calibrated and validated in regular intervals. Instead of manually checking the sensors, it has proven useful to provide redundant instrumentation within the data center infrastructure. For example, when monitoring electrical energy delivered to a computer, the amount of heat energy removed from that computer should be monitored as well. Due to the principle of energy conservation, the amount of electrical and thermal energy will be equal. In case the measured values differ by more than what is expected due to the measurement's tolerance, further investigation into the involved sensors has to take place.

1.3 OPTIMIZED HPC SYSTEM HARDWARE

According to the Intel® WCP Family Power Budget and Thermal Config Tool, the power consumption in an Intel® server system which is comparable to the hardware of a compute node in the SuperMUC Phase II supercomputer [23] can be estimated as outlined in Table 2. This means that energy efficiency in today's supercomputers is mostly determined by the power consumption of the central processing units (CPUs).

Like all computer chips, the dynamic power consumption P of a CPU can be approximated as

$$P = CV^2 f \tag{4}$$

with:

- C: capacitance of the chip
- V: operating voltage
- f: operating frequency

For the design of the power delivery paths to the CPU and the CPU's cooling infrastructure, CPU manufacturers define the Thermal

Processor	*Year*	*Process*	*Base Frequency*	*Voltage Range*	*TDP*
Pentium® II Xeon®	1994	250 nm	400 MHz	2.0 V	18.6 W
Xeon® 1.4	2001	180 nm	1.4 GHz	1.75 V	64 W
Itanium® 2 1600	2004	130 nm	1.6 GHz	n/a	122 W
Xeon® 2.8	2004	90 nm	2.8 GHz	1.2875 V-1.4125 V	103 W
Xeon® 5030	2006	65 nm	2.66 GHz	1.075 V-1.35 V	95 W
Itanium® 2 9040	2007	90 nm	1.6 GHz	1.0875 V-1.25 V	104 W
Xeon® E5405	2007	45 nm	2.0 GHz	0.85 V-1.35 V	80 W
Xeon® W3670	2010	32 nm	3.2 GHz	0.8 V-1.375 V	130 W
Xeon® E5-2680	2012	32 nm	2.7 GHz	0.6 V-1.35 V	130 W
Xeon® E5-2697v3	2014	22 nm	2.6 GHz	0.65 V-1.30 V	145 W
Xeon® Gold 6140	2017	14 nm	2.3 GHz	n/a	140 W

Table 3: History of selected parameters in different Intel® server processors. Source: http://ark.intel.com/

Design Power (TDP), which describes the maximum average power under full load. Table 3 shows a comparison of selected Intel® server processors over time with respect to their power-consumption related properties. As the table shows, chip manufacturers have managed to scale down their semiconductor manufacturing processes to allow for smaller transistors. This reduces the chip's capacitance and due to the smaller gate sizes, lower operating voltages are sufficient to switch the transistors at the same speed. According to Equation 4, both effects lead to a reduction of power consumption as semiconductors shrink. Despite these optimizations of the power consumed per transistor, the thermal design power of CPUs has increased over time since CPU manufacturers have stuck to the exponential growth of the number of transistors per CPU according to Moore's law.

To compensate for the increased TDPs, all processor manufacturers nowadays provide techniques that allow for adapting processor performance according to demand using dynamic voltage and frequency scaling (DVFS). This means that a CPU can change its operating frequency and supply voltage dynamically according to the application's needs. The power consumption of the chip thus varies accordingly.

Finally, chip makers make extensive use of power and clock gating which allows for shutting down entire functional units on a chip or at least skip the supply of the clock signal in order to prevent any logic gates from switching. Through the use of these technologies, the power consumption of a CPU and subsequently of the entire computer can vary significantly between idle and full load: An idle node of the type mentioned in Table 2 consumes about 50 W of power.

Also, with the ability to adjust DVFS, it becomes possible to trade in application performance for power consumption. It is therefore important to equip HPC systems with thorough and precise power monitoring capabilities that keep track of power and energy usage along with other relevant metrics (i.e. CPU frequency). With this information at hand, HPC system users and operators may tune the system for individual applications.

1.4 OPTIMIZED HPC SYSTEM SOFTWARE

The goal of the HPC system software stack is to provide workload management and to configure the system according to given policies using the hardware tuning and monitoring capabilities. Many well-established software solutions exist that perform workload management for optimal resource utilization (e.g. PBS, LSF, Slurm. . .). Also, many software solutions exist for monitoring the system's state and for analyzing the system's performance, which will be discussed in greater detail in Chapter 3. However, when it comes to energy awareness in system management software, existing solutions today only allow for reporting the energy consumed by an application and provide features to restrict the power consumption of a supercomputer in order not to exceed a predefined maximum (power capping). As of today, the Leibniz Supercomputing Centre is one of few HPC centers that uses energy aware resource management software. On the SuperMUC supercomputer [23], LoadLeveler is used to adapt the processor frequencies according to the applications' needs [13].

The area of HPC system software has large potential for future developments in the field of energy efficient HPC. Only at the system management software level, a global view of the many parallel processing units of a supercomputer is available. Thus, proper orchestration of a HPC system's energy-related tunables can only happen here.

Furthermore, future HPC system software can have links from and to the building infrastructure management software or even to the utility providers allowing to optimize globally across all pillars of the 4-pillar model. This facilitates building a supercomputing center, which increases its usage and power consumption whenever there is sufficient power available in the grid (i.e. prices are low) while lowering usage and power consumption during times when less power is available and prices are high.

1.5 OPTIMIZED HPC APPLICATIONS

Optimizing HPC applications for energy efficiency is a straightforward task because, in almost all cases, optimizations for application performance also yield optimizations in energy efficiency.

The theory behind this claim is as follows: energy is the integral of power over time and optimizing the application performance aims at reducing the execution time of a program. Thus, reducing the execution time of a program automatically reduces the energy, unless the power consumption increases at the same time to the same extent. However, the power consumption of a computer has both, a lower boundary (i.e. the power consumption when the system is idle) and an upper boundary (i.e. the maximum thermal design power of the system). Thus, for any HPC program that already causes the machine to run at or near its maximum power, the power consumption cannot increase. Optimizing the runtime of such program will therefore result in a reduction of its energy consumption.

As a consequence, energy aware application developers should critically review their algorithms, avoid load imbalances in their parallel codes, and rely on numerical libraries that are optimized for their systems. Additional software development techniques that specifically target the reduction of the energy consumption of algorithms such as significance based computing [24] are only emerging and beyond the scope of this thesis.

1.6 CROSS-PILLAR OPTIMIZATIONS

Like other supercomputing centers, the Leibniz Supercomputing Centre of the Bavarian Academy of Sciences and Humanities has an organizational structure that resembles the 4-Pillar model in that there are organizational units for the building infrastructure, hardware operations, system software, and application support. And, it is safe to assume that the techniques for optimizing the energy efficiency within their scope are well known within each team. However, besides these optimizations within each pillar, a global view or, ideally, a global optimization scheme is required that acts across all pillars. If data center operators fail to do so, optimizations performed within one pillar may have negative impacts on the energy efficiency within another pillar.

A prominent example for this effect is the common recommendation to raise the computer room temperatures. At first, this approach may sound promising since a higher computer room temperature results in reduced chiller activity and longer periods in a year in which chiller-less cooling is possible. However, although the new temperature in the room might still conform to the specifications of the IT equipment, servers, power supplies and other components with built-in fans might increase their fan speeds as a reaction to increased temperatures of their air inlets. Since the power consumption of a fan is a cube function of the fan speed and airflow, increased fan speeds will contribute measurably to the electricity bill.

Another example is related to the reuse of excess heat. At LRZ, heat produced by HPC systems is being used to heat the office buildings in winter. As an attempt to make use of the heat during summer time, LRZ has deployed two prototypes (CooLMUC 1 & 2) that use adsorption refrigerators in which the excess heat drives a chiller process. As a consequence of the second law of thermodynamics, energy can only be recovered from a warm medium during the process of transferring the heat from the warm reservoir into a cold reservoir. The *Carnot theorem* defines an upper bound for the efficiency of this process:

$$\eta_{max} = 1 - \frac{T_C}{T_H} \tag{5}$$

with:

- η_{max}: maximum efficiency of the excess heat reuse process
- T_C: temperature of the cold reservoir (in Kelvin)
- T_H: temperature of the hot reservoir (in Kelvin)

The theorem shows that the efficiency of every process driven by excess heat is bound by the difference between T_C and T_H. The only recooling medium that is commonly available in unlimited form is outside air. This means that T_C is determined by the outside air temperature and as such beyond engineering control. The only way to increase the efficiency of the heat reuse process is to increase T_H. In the context of supercomputer waste heat, this means that the outlet temperature of the computer cooling infrastructure has to be increased. Increasing the outlet temperature can be done by either increasing the inlet temperature or by reducing the volume flow of the coolant. In both cases, the compute equipment that has to be cooled will run at higher temperatures. Equation 4 gave an approximation of the dynamic power consumption of a processor, i.e. power that is spent to toggle the transistors within the chip. However, in addition to that dynamic power, computer chips consume a static amount of power (leakage power) that mainly depends on the semiconductor manufacturing process and the chip temperature: higher temperatures result in higher leakage power. Joining the leakage power challenge and the efficiency challenge of the heat reuse process, one ends up with an optimization problem: On one hand, higher temperatures will improve the heat reuse process. On the other hand, higher temperatures come at the cost of higher power consumption of the computer. The further analysis if this problem on the CooLMUC 2 supercomputer can be found in [25].

Unfortunately, the extent to which the operating temperature influences the leakage power of a computer is hard to predict upfront as it is related to the semiconductor manufacturing process and the chip's internal voltages among others [26]. Thus, the described optimization problem can only be solved through practical experiments which span building infrastructure components (heat reuse process), HPC system hardware (leakage power characteristics), and system software (monitoring energy balance).

Both examples in this section show that a cross-pillar view is necessary for further optimizing the energy efficiency of HPC data centers. This thesis covers a fundamental requirement of generating such cross-pillar views: a scalable system monitoring infrastructure capable of integrating sensor information from sources across all pillars.

2 METRICS FOR ENERGY EFFICIENCY IN HPC

The 4 Pillar Framework is a useful tool to map various activities in improving the energy efficiency in HPC. It can be used to identify the respective stakeholders and to manage cross-pillar interactions. However, it does not provide means of quantifying the success of the actions taken. For this purpose, different metrics have gained varying popularity.

2.1 FLOPS PER WATT

When it comes to HPC, the Flops per Watt metric is still one of the most widely used approaches to describe the energy efficiency of a HPC system. As the name indicates, it puts into relation the achieved sustained number of floating point instructions per second (Flops) with the power consumption of the machine during the run. A widely accepted benchmark to use as a measurement for Flops is the High Performance Linpack (HPL) benchmark that is used for the Top500 list. The resulting Green500 list [27] ranks the systems listed in the Top500 list according to their Flops per Watt ratio. With 22 releases to date (as of November 2017), the Green500 is a widely accepted ranking and energy efficiency in terms of Flops per Watt has become a deciding criteria for many supercomputer procurements.

Criticism related to the metric centers around the use of HPL as underlying benchmark and the vague requirements regarding the measurement granularity and accuracy. HPL is a benchmark that solves a large system of linear equations using LU decomposition. As such, it is easy to parallelize and mainly targets the performance of the CPUs while other factors influencing supercomputer performance such as network or I/O performance are not honored adequately. It also largely benefits from single instruction multiple data (SIMD) style architectures. While this has been seen as a typical reference case for HPC applications years ago, modern HPC applications are far more complex causing an increasing gap between system performance according to HPL and the performance of real-world application codes.

The second aspect that is often criticized concerning the Flops per Watt metric and the Green500 list is related to measurement quality. In the original power measurement tutorial published in 2007 [28], only few requirements were given regarding the measurement procedure and quality. The underlying assumption at the time was that supercomputer power consumption is homogeneous across different nodes and over time during the same workload. Thus, the original rules allowed for measuring only a fraction of the system during a relatively small time interval of the HPL run.

With the increasing use of power measurement equipment in supercomputers, it became evident that HPL power consumption varies significantly over time. Therefore, the Energy Efficient High Performance Working Group (EE HPC WG) [29] initiated a discussion on improving the measurement methodology. The resulting guidelines [30] address the weaknesses of the original measurement tutorial while still recognizing that supercomputing sites and systems feature different types of measurement instrumentation with varying degrees of measurement quality. Thus, a key aspect of the new guidelines is the definition of three levels of measurement quality ranging from level 1 ("adequate") via level 2 ("moderate") to level 3 ("best") depending on aspects such as the measurement granularity, machine fraction instrumented, and the list of subsystems included in the measurement. The EE HPC WG methodology is under constant refinement using a transparent change management process.

Despite the success of the EE HPC WG's efforts in raising awareness in the field of accurate power measurement instrumentation, most systems in the Green500 provide a level 1 reading only. This may, at least partly, be due to the fact that level 1 measurements typically yield better Flops per Watt results than level 2 or level 3 measurements [31].

2.2 POWER USAGE EFFECTIVENESS

As opposed to the Flops per Watt metric, which targets the energy efficiency of a supercomputer system, Power Usage Effectiveness (PUE) [32] is a metric for entire data centers. It was introduced by The Green Grid [33], an industry association promoting resource and energy efficiency in the information and communication technology sector. PUE relates the total energy consumption of a data center to the energy consumption of the IT equipment:

$$\text{PUE} = \frac{\text{Total Facility Energy}}{\text{IT Equipment Energy}} \tag{6}$$

IT Equipment Energy is the amount of the energy spent on servers, storage and networking equipment. *Total Facility Energy* comprises of the IT Equipment Energy and all other energy consumed in the data center such as cooling equipment, power delivery equipment, lighting, heating, and many others. Due to this, the theoretical best PUE for a data center is 1.0 indicating that there is no energy being spent for the data center building infrastructures and all energy is being consumed by the IT equipment only.

At the time of introduction in 2006, many data centers would exhibit a PUE of more than 2. However, due to the strong promotion of PUE by The Green Grid, the necessity to lower the infrastructure overhead in data centers is now widely known among data center operators and modern data centers can be designed for PUE values of 1.5 or better.

Although the definition of PUE is clear, PUE is challenging to measure in practice, mostly due to a lack of availability of measurement points. While this may not be problematic for keeping track of improvements within someone's own data center, any attempt to compare the PUE values of different data centers is unlikely to withstand a closer examination.

Another common mistake related to PUE is to use it as an encompassing metric for data center energy efficiency. This is wrong because PUE makes no statement regarding the efficiency of the IT equipment itself (e.g. Flops per Watt): putting an old supercomputer into a brand new data center has the potential of a low PUE, yet the overall energy efficiency might have gotten even better by renewing the machine instead of the data center.

Finally, optimizing solely for PUE might introduce inefficiencies elsewhere. Let's reconsider the cross-pillar optimization example from Chapter 1: the computer room temperature was raised to save power in the data center infrastructures, yet the increased fan speeds within the computer hardware caused an increase in IT power consumption. For the PUE value, this shift from cooling power to IT power is a clear benefit, despite the uncertainty regarding the overall benefit from the approach. To address this problem of PUE, Patterson et al. introduced TUE [34], which restricts the IT Equipment Energy part of the equation to the components within the IT equipment that contribute to the computation or storage (CPUs, memory. . .). Unfortunately, this makes TUE even harder to measure, because the detailed power instrumentation required for such measurement can only be found in systems, which are specially designed for this purpose.

2.3 ENERGY TO SOLUTION

From a supercomputer operator's perspective, neither Flops per Watt nor PUE are fully satisfactory. The goal for supercomputing centers is to maximize the scientific value of the computations while minimizing computation costs. This implies that the computing resources have to be used effectively, generating as many results with high impact for the HPC users in their respective research domain as possible during a system's lifetime. Unfortunately, none of the two metrics Flops/Watt and PUE, nor their combination satisfy this need. The costs for running supercomputing applications comprise mainly of the application development costs (typically beyond the scope of the supercomputing center), personnel costs at the supercomputing site, machine write-off costs, and electricity costs. Assuming that the maximization of the scientific output is in the natural interest of the supercomputing users and restricting the total costs to their energy related parts yields *Energy To Solution*: the amount of energy spent to execute a given supercomputing application to deliver a scientific computation's result.

Energy to Solution is a metric that allows for comparison between different supercomputers and supercomputing centers and, thus, it is also suited for rating the overall energy efficiency of a supercomputing center for a given application.

A challenge, however, is to accurately measure Energy to Solution. While a fraction of the total energy consumed by a supercomputing application is easy to measure (i.e. the sum of energy consumed by all computing nodes involved in the calculation), other components such as background storage, or networking gear, as well as infrastructure components such as chillers, cooling towers, or pumps can be shared among multiple systems. In these cases, a true measurement to derive Energy to Solution is impossible. One approach to mitigate this is to split up the energy consumed by such shared devices using approximative models that try to derive the fractional energy consumption per supercomputing application. To attribute the networking energy consumption to multiple programs running in parallel on the same supercomputer over a shared network, one could, for example, use the number of networking packets sent and received per application as a base metric from which the fraction of energy consumed in the networking equipment per program is derived.

The challenge of shared resources is likely the reason why all commercial solutions for monitoring per-job Energy to Solution in HPC solely rely on node-level energy measurements and ignore shared components such as storage, networking, and the building infrastructures.

2.4 OTHER METRICS

In addition to the aforementioned metrics, one may consider other metrics as relevant for the assessment of energy efficiency in HPC:

ENERGY REUSE EFFECTIVENESS (ERE) is an extension to PUE that allows for subtracting any energy from the total facility energy that is being reused, e.g. for heating. Therefore, as opposed to PUE, ERE may take a value smaller than 1.0.

WATER USAGE EFFECTIVENESS (WUE) is another metric developed at The Green Grid. It relates the use of water to the total facility energy. Considering that in some geographical locations, water is a scarce resource, the approach to evaporate water in order to increase cooling tower efficiency may be unfavorable. And even in regions with sufficient access to water, pre-treatment of the water (e.g. de-hardening) before evaporation also requires energy.

DATA CENTER WORKLOAD POWER EFFICIENCY (DWPE) combines Performance per Watt for arbitrary workloads with a system and data center specific PUE [35]. With this, DWPE is highly encompassing and honors that depending on the type of workload, certain computer architectures might be more efficient than others and that the same system will behave differently when being operated in another datacenter.

.

An observation that holds for all metrics related to energy efficiency in HPC is that the actual assessment of the metrics requires a combination of multiple data sources, often over long time periods. Therefore, the following chapter analyzes the current state in system monitoring and performance analysis for capturing, storing, and processing information from system sensors that are necessary to derive above metrics.

3 THE HPC SYSTEM MONITORING CHALLENGE

System monitoring for HPC imposes a multifaceted challenge. While for some purposes the tools originating from the commercial server domain may be applicable to HPC systems, the size of HPC systems and the stronger need for detailed insight into application performance often require special solutions for HPC. Traditionally, system monitoring activities can be divided into two parts:

LIVE CHECKING comprises of all means to periodically check the availability of a system or its services.

TIME SERIES MONITORING consists of all activities related to collecting, analyzing, and storing information related to various operational metrics of the system over time.

Time series monitoring information for IT systems covers data indicating the physical conditions of the machine (i.e. temperatures, fan speeds, bit error rates...), application performance data (cycles per instruction, number of cache misses...), and in many recent systems also power consumption information. Means of access to system monitoring data for generating time series can be grouped into two categories:

IN-BAND ACCESS exposes sensor information to the system software and possibly also to the applications running on the compute hardware. On the hardware level, data is transferred from the sensors to the CPU for in-band access via PCIe, I^2C, SPI, or other busses. The OS kernel accesses these busses using memory mapped I/O or machine specific registers (MSR) and exposes the data to the user via procfs, sysfs, or other interfaces.

OUT-OF-BAND ACCESS relies on external monitoring systems and separate read-out paths that do not interfere with the system software and compute hardware executing the workload. For this purpose, server systems provide baseboard management controllers (BMCs) that operate independently from the main compute hardware on a node. In most cases, dedicated monitoring servers access the BMCs through a distinct management network using special protocols like the Intelligent Platform Management Interface (IPMI) to collect the sensor information.

3.1 EXISTING SOLUTIONS FOR STANDARD SYSTEM MONITORING

Standard System Monitoring consists of the activities and tools that enable a system's administrator to

- observe the physical operating conditions of an IT system,
- measure and analyze simple performance metrics over time, and
- be alerted in case of system hardware errors or unavailability of services.

This is a common requirement in the area of IT infrastructures and services that operate 24/7. Thus, many software tools exist in this domain. Since the goal of these tools is only to provide the system administrator an overview over the general health condition of their hardware and services, the spatial and temporal resolution of these tools is coarse: monitored entities are restricted to the most relevant ones and data is acquired in intervals of minutes. Only in case of anomalies, it is expected that the system administrator performs a closer inspection of the affected entities to identify any potential issues.

This section introduces the most widely used tools for standard system monitoring today: *Nagios*, *Icinga*, and *CheckMK*.

3.1.1 *Nagios*

Nagios [36] is considered the de-facto standard for IT systems monitoring. It is released as open-source software under the GPL v2 license backed with commercial support by Nagios Enterprises LLC [37].

In Nagios, the "Nagios Core" implements the basic functionality such as an inventory of devices (servers, switches. . .) to be monitored. Through a variety of publicly available "Nagios Plugins", the Nagios Core can trigger live checks on network hosts (host check) or network services (service check) and collect time series data from various sources. Due to its plugin based approach, users can easily extend Nagios with their own plugins for customization. In addition to these active checks (i.e. checks triggered by the Nagios Core), Nagios supports passive checks in which external processes can implement and trigger host or service checks at their own discretion and submit their result to the Nagios Core.

Although Nagios can run several checks in parallel by forking from the core process for each check, the centralized approach of the single Nagios core imposes a limit on scalability in large IT environments. To mitigate this problem, Nagios implements a dependency logic, which allows to model links between services and hosts. In addition to performing checks at regular intervals, this logic can trigger checks on-demand whenever a host or service status changes. System administrators may therefore configure Nagios to only check certain services at regular intervals and have Nagios check the hosts associated with a service only when the service status changes. In essence, this feature can be considered as a lightweight implementation of a CMDB[1].

While reducing the load on the Nagios core through dependent checking is possible and effective for live-checking applications, it cannot be used for collecting time series data in which all data points are to be kept for later analysis. Thus, the base Nagios feature set for dealing with time series data is limited, albeit various Nagios plugins exist that store and visualize time series data through the help of RRDTool (see Section 3.1.4).

3.1.2 *Icinga*

Icinga [39] originally started as a fork of Nagios. The Icinga project integrated a series of patches originating from the community and provided a more modern graphical user interface than Nagios. Meanwhile, Icinga 2 has been released. It has been written from scratch and aims at modernizing the monitoring core. A notable improvement in Icinga 2 is the support for distributed monitoring and high availability features. Still, Icinga 2 remains compatible with the Nagios plugin architecture for providing host and service checks and thus suffers from the same performance limitations as Nagios when it comes to executing these checks from a single monitoring node.

Similar to Nagios, storing and visualizing time series data is done using RRDtool.

3.1.3 *Check_MK and Open Monitoring Distribution*

The scalability challenge in Nagios and Icinga lies in the centralized core, which has to trigger all active checks. Even though the checks are forked out and may run as separate processes in parallel to the core, the design results in a single server having to initiate many connections to perform the host and service checks.

The Check_MK [40] project started as a plugin to Nagios which improves the case in which the Nagios core has to check for multiple services on the same node. Instead of checking the node's services one after another, Check_MK relies on a node-side script that performs all the checks locally and generates a summarized report containing all node status reports. This summarized report can then be transferred

1 CMDB: Configuration Management DataBase – according to IT infrastructure guidelines such as ITIL® [38] or ISO 20000, a configuration management database models all configuration items (hardware, services, applications. . .) and their relationships.

in a single connection, thus freeing the Nagios core from the effort of having to initiate at least one connection per service.

Assuming a properly crafted node-side script to generate the summarized report, Check_MK can automatically detect all services running on a node and configure Nagios automatically. This feature proves very powerful in environments where nodes are re-provisioned frequently and the need of adapting the Nagios configuration upon each re-provisioning becomes superfluous.

The Check_MK team also develops Multisite, another web-based graphical user interface to Nagios. Multisite integrates seamlessly with the Check_MK approach of auto-configuring service checks and makes the monitoring setup including users, groups, roles, and access rights configurable through the web browser interface.

Since a full setup of a monitoring server from scratch including Nagios, Check_MK, and Multisite can become cumbersome due to the numerous dependencies on external libraries and tools, the authors of Check_MK combine all their tools including third party dependencies into a single distribution, the Open Monitoring Distribution (OMD). With OMD installed, administrators can set up entire Nagios/Check_MK installations with a single command line operation.

Although Check_MK improves the scalability of Nagios further, it still relies on the central Nagios core for collecting all monitoring information, implying that the number of monitored hosts can not be extended indefinitely. Also, for storing and visualizing time series data, Check_MK sticks to the use of RRDtool.

3.1.4 *RRDTool*

Due to its use in Nagios, Icinga, and many other applications, the Round Robin Database tool [41] is probably the most widely used tool to store and visualize time series data. A few key features of RRDtool explain its popularity for system monitoring:

THE ROUND ROBIN STORAGE scheme with pre-allocated data files causes the RRDtool disk usage to stay constant ensuring that the monitoring servers do not run out of disk space.

AUTOMATIC AGGREGATION for older data allows for keeping averaged or otherwise aggregated values at lower temporal resolution allowing to trade in resolution with storage duration under constrained disk space.

A POWERFUL GRAPHING TOOL for generating time series graphs of the data stored in the round robin database files.

RRDtool is licensed under the GNU General Public License and ships as a set of command line utilities. Support for the Common

Gateway Interface (CGI) ensures seamless integration of up-to-date time series graphs into web based graphical user interfaces.

RRDtool works entirely file based and thus relies on the underlying file system for implementation details related to concurrent access and locking. While this simplifies the implementation of RRDtool significantly, it means that any use of RRDtool in a larger environment with shared file systems is automatically limited by the locking and metadata performance of the file systems. Since all operations on the RRD databases require the execution of a command line program that goes through the cycle of opening, locking, writing or reading, and closing the file, all RRDtool operations impose a significant overhead over other database management systems that run continuously as service daemons allowing for caching data in memory until committing a full batch of data to background storage in one single operation. This, in conjunction with the fact that RRDtool uses Unix epoch time stamps internally (with a temporal resolution of 1 second), makes RRDtool unsuited for storing high-frequency time series data.

3.2 PERFORMANCE ANALYSIS TOOLS

In high performance computing, as the name implies, application performance plays an important role and is therefore subject to thorough analysis. For this purpose, processors are equipped with special registers, called hardware performance counters. Processors can be configured to use the hardware performance counter registers to count the occurrence of certain events such as retiring of instructions or cache misses.

The combination of these hardware performance monitoring capabilities with other software performance analysis techniques, such as sampling based call graph analysis or source code instrumentation allow for an in-depth analysis of the performance behavior of an application. Over the years, several tools have evolved that collect and visualize application performance data to assist HPC application developers in further optimizing their codes.

Since performance analysis tools aim at closely correlating the collected data with the execution of the application, performance analysis tools traditionally collect performance data in-band. Thus, conducting application performance analysis always comes with a performance impact on the running application and developers of performance analysis tools strive to minimize that impact.

This section briefly introduces some of the most common tools for application performance analysis. *Scalasca*, *Vampir*, *Periscope*, and *Tau* are based on the common *Score-P framework* for collecting application performance data. *HPCToolkit*, *ARM MAP*, and *Intel® VTune™ Amplifier XE* ship with their own implementation for application data collection.

3.2.1 *Score-P Based Tools*

Score-P ("Scalable Performance Measurement Infrastructure for Parallel Codes") [42] is the underlying data acquisition layer for a variety of performance analysis tools. It supports various HPC programming paradigms ranging from traditional MPI and OpenMP to CUDA and OpenCL.

Score-P works by instrumenting the code at compile time with calls to a library that collects event trace data into trace files using the OTF2 (Open Trace Format 2) file format or call-path based profiling data using the Cube 4 profiling data format. Writing the application performance data to files using a standard file format allows for doing performance analysis of the exact same run using different tools, meaning that application developers do not need to re-run applications multiple times under different performance analysis tools. In addition to writing application traces and application call-path profiles to files, Score-P also provides an online interface over TCP/IP.

The following tools rely on Score-P for performance data acquisition:

SCALASCA [43] [44] is a BSD licensed performance analysis tool, developed at the Jülich Supercomputing Centre, Technische Universität Darmstadt, and the German Research School for Simulation Sciences. It is designed particularly for large-scale systems breaking down the collected data along a 3-dimensional performance space: $(M \times P \times S)$ where

M represents the set of performance metrics of interest

P represents the set of program locations (e.g. functions)

S represents a view of the system (e.g. set of nodes)

The Cube viewer (Scalasca's graphical user interface) helps identifying performance issues by implementing a colored scheme that quickly helps identifying the spots in the performance space with the largest room for improvement.

VAMPIR [45] is a graphical performance analysis tool developed at Technische Universität Dresden. It displays the collected application performance data either as time series or in statistical summary charts.

PERISCOPE [46] is a performance analysis and tuning framework developed at Technische Universität München. Instead of doing post-run analysis of the data sets generated by Score-P, it uses the online interface of Score-P to adapt and refine the set of monitored properties during the application run. An Eclipse plugin can be used for visualizing the generated data.

TAU (Tuning and Analysis Utilities) [47] [48] is a performance analysis toolkit developed at the University of Oregon. The profiling capabilities of TAU are comparable to other performance analysis tools, including various options for generating graphs from the performance data. In addition, however, TAU also provides a code analysis package that performs static code analysis.

3.2.2 *HPCToolkit*

HPCToolit [49] [50] is a BSD-licensed performance analysis package developed at Rice University. It uses statistical sampling to derive its performance metrics and, thus, does not require applications to be recompiled for profiling. Nevertheless, if required by the application developer, HPCToolkit provides an API to interface with the sampling process from within the application.

In addition to these profiling features, HPCToolkit supports the user in correlating the application's binaries with the original source to understand the optimizations performed by the compiler (function inlining, loop unrolling, vectorization. . .). This is particularly useful in identifying code sections with degraded performance due to missing optimizations of the compiler, despite the application developers wrongly assuming the compiler to be able to optimize the given section.

Besides providing standard metrics from hardware performance counters and other common sources, HPCToolkit also allows for defining "derived metrics" which allows the user to define arbitrary arithmetic expressions that combine existing metrics into new ones.

HPCToolkit also provides a graphical user interface that helps in visualizing performance metrics either based on code and call tree or as time series.

3.2.3 *ARM MAP*

ARM MAP [51] (*Allinea MAP* in days preceding the acquisition of Allinea by ARM Ltd.) is a commercial profiling tool that is part of the ARM Forge tool suite. Similar to HPCToolkit, it performs application profiling by statistically sampling the running application. Therefore, it also does not require the application to be recompiled or otherwise annotated.

A recent feature in ARM MAP is the ability to correlate application performance with power consumption. For this, ARM MAP supports various sources for power or energy measurements, such as the Intel® RAPL interface.

3.2.4 *Intel® VTune™ Amplifier and Trace Analyzer & Collector*

Intel® VTune™ Amplifier [52] in conjunction with the Intel® Trace Analyzer & Collector form another set of commercial profiling tools.

Data acquisition is performed using statistical sampling similarly to HPCToolkit and ARM MAP. VTune™ Amplifier covers node-level analysis including OpenMP parallel codes, whereas the Trace Analyzer hook into the MPI layer to profile MPI communication.

Similar to ARM MAP, these tools now also support the correlation of application performance with power consumption data.

3.3 LIMITATIONS OF EXISTING SOLUTIONS

As shown in this chapter, a multitude of tools exists for system and performance monitoring and analysis. Regarding the requirements for calculating metrics described in Chapter 2 such as Energy to Solution, however, limitations of these tools become apparent. All aforementioned tools suffer from a limitation in scope in that they do not natively support the integration of information from the building infrastructure domain. Additionally, while the system monitoring tools can be configured to monitor multiple supercomputing systems, meaning that they can have a global system view, they do not have an understanding of the temporal allocation of supercomputer resources to applications or users. On the other hand, performance analysis tools work on a per-application and per-user basis, but lack a more global view across applications or even systems.

While in theory, application performance data captured by Score-P or other performance analysis tools could be fed into the system monitoring solutions, the limited scalability of the Nagios Core or RRDTool would render such an attempt impractical. Additionally, the performance impact of an always-on performance monitoring infrastructure would reduce throughput and increase operational costs beyond feasibility.

3.4 ADDRESSING THE LIMITATIONS

Due to the limitations of existing system and performance monitoring solutions, research at the Leibniz Supercomputing Centre has yielded approaches to address these limitations. The basic goal of these efforts is to remove the spatial and temporal restrictions of existing tools

- by integrating IT and building infrastructure monitoring data,
- by accessing more information sources and sensors, and
- by researching ways of efficiently organizing data for improved scalability.

3.4.1 *Integrated Monitoring with PowerDAM*

PowerDAM [53] is LRZ's software for integrating conventional supercomputing system monitoring with building infrastructure monitoring and resource management. It collects sensor data from various sources spanning multiple HPC systems and LRZ's building infrastructure management systems for cooling and power distribution. Data is collected from all sources once per minute and stored in a MySQL database.

With the integration of building infrastructure data, PowerDAM is capable of providing live PUE information for SuperMUC, LRZ's Tier-0 supercomputing system. In addition, the connection to LRZ's resource management tools LoadLeveler and Slurm, enables Energy to Solution measurements for each application or summarized energy consumption reports per user. For shared resources such as cooling and networking, PowerDAM can split the energy consumed of the shared resource according to configurable parameters.

PowerDAM has large potential in addressing the limitations of traditional system monitoring solutions when it comes to assessing the supercomputing center's energy efficiency. However, relying on a single, centralized database introduces inherent scalability limitations effectively limiting the further scale of PowerDAM to include more systems or increase sampling frequency for sensor data.

A tool similar in functionality to PowerDAM is DataHeap [54] developed at Technische Universität Dresden.

3.4.2 *Smart Data Acquisition*

Tackling the scalability challenge for systems collecting vast amount of data can be done in various ways. One of the smartest concepts is to avoid the storing of unnecessary data.

The PerSyst tool [55] developed at LRZ provides always-on performance monitoring for all applications running on LRZ's supercomputers. To minimize its overhead, PerSyst acquires performance monitoring data every 10 minutes for a 30 second period as research has shown that this interval is sufficient for a general overview of the performance behavior of an application. To additionally limit the amount of storage space required for the collected data, PerSyst features a smart data acquisition scheme. This scheme causes PerSyst to start with monitoring only high-level performance properties (e.g. total cache misses). In case one of the monitored properties exceeds a configurable threshold, PerSyst automatically refines the measurement to include additional properties (e.g. level 1 and level 2 cache misses). Through this concept, data is only stored when it is relevant due to an excursion over the predefined threshold. For fully optimized applications, only a minimal set of data is stored at all whereas for applications with performance problems, all relevant properties are kept for later analysis.

3.4.3 *Compressed Data Storage*

In addition to only store relevant data, PerSyst also implements a compression scheme for its data. Instead of storing the raw values for each property on each node, PerSyst collects the per-property data from all nodes, sorts the values in ascending order and only stores the values of the deciles across the resulting distribution. Within the PerSyst database, the deciles are associated with the accounting information of the running application. Since key aspects of the application's behavior are directly visible from the distribution function, the stored data is still sufficient to generate meaningful reports for application developers despite the "lossy compression" employed on the performance monitoring data.

3.4.4 *Distributed Data Storage*

Unfortunately, the concepts of PerSyst are only partially applicable to PowerDAM. As opposed to CPU performance counter data, power monitoring and other system monitoring data (voltages, temperatures, fan speeds. . .) is often requested in its full time series per sensor. Also, omitting data at the time of acquisition requires upfront knowledge on the expected values and dependencies between different data sources. Since data center wide integrated monitoring for energy efficiency is a relatively new field of research, little is known about the dependencies and interactions. It therefore makes sense to collect as much data as possible for later analysis.

With the centralized database being the bottleneck in PowerDAM, the best approach to solving the scalability problem is to move to a distributed database. Of particular interest in this context are NoSQL database systems that give up on the traditional atomicity, consistency, isolation, and durability (ACID) properties of traditional relational database management systems as the loosening of these properties allows for distributed database systems that exhibit performance scaling almost linear to the number of database nodes.

The following chapter introduces Data Center DataBase (DCDB), a framework that builds on the distributed Cassandra NoSQL database for scalable collection of system monitoring data.

4 THE DCDB FRAMEWORK

The Data Center DataBase (DCDB) framework is a concept and supporting software toolset for building highly scalable time series monitoring infrastructures. DCDB can integrate an almost arbitrary number of sensors from a multitude of different sources with a temporal resolution of up to 1 nanosecond. In the design of DCDB, scalability has always been first priority. The following aspects of DCDB contribute to its scalability:

The Apache Cassandra NoSQL database has a very restrictive data model and loosens ACID compliance resulting in a database management system that exhibits nearly linear scalability when increasing the number of database servers.

A push model based on the lightweight MQ Telemetry Transport (MQTT) protocol is used instead of the traditional pull models for acquisition of sensor data. This eliminates the need for a central server that initiates sensor readings at regular intervals and allows for a *change of value* approach for updating sensor values in addition to classical periodic updates.

Local storage – global analysis means that DCDB promotes setups in which the time series data is kept relatively close to its source. E.g. supercomputers could operate one database node per rack that stores this particular rack's sensor data, avoiding the congestion of networks with high frequency monitoring data. Due to the distributed nature of the database, analyzing the data across multiple database nodes, however, is no different from analyzing data from a single database node. In applications such as system monitoring, where the number of database inserts often outweighs the number of reads, this approach significantly lowers the amount of monitoring related network traffic.

4.1 DCDB overview

Figure 3 provides a high level overview of the DCDB data acquisition and storage process.

Sensor data originating from a variety of sources is sent into DCDB via the MQTT protocol. A Collect Agent receives each sensor reading and, if not already done at the data source, associates a time stamp

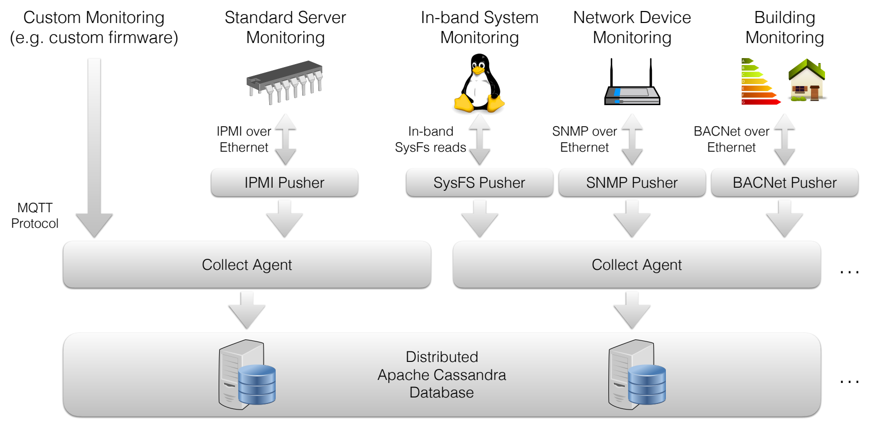


Figure 3: Overview of the sensor data acquisition and storage process in the DCDB Framework.[1]

to it. The Collect Agent then stores the data in one of the Cassandra database nodes.

Whenever needed, additional instances of Collect Agents and database nodes can be configured to distribute the load of incoming data.

To avoid any loss of information that may result from early modification of sensor data (e.g. scaling, averaging...), users of DCDB are encouraged to push sensor data into DCDB in its raw form. DCDB facilitates the handling of data in raw formats by providing built-in means of scaling the data and converting units. Through the definition of *virtual sensors*, users can create additional data series that combine data from multiple sensor sources.

For retrieving the stored data, DCDB provides the `dcdbquery` command line utility. Using the `dcdbplot` utility, this data can be turned into time series graphs. All DCDB command line utilities rely on the shared `dcdblib` library which may also be used to develop custom applications that work with the data stored in DCDB. All functions provided by dcdblib will be referred to as *DCDB API*.

4.2 APACHE CASSANDRA NOSQL DATABASE

Apache Cassandra [56] is a cross-platform, distributed database management system. It is written in Java and licensed under the Apache License. Cassandra development is managed by DataStax, Inc. [57], a Santa Clara based company that also provides commercial releases with additional features and commercial Cassandra support.

1 The image of Tux, the official Linux mascot used in this figure is artwork by Larry Ewing and The GIMP.

4.2.1 *History*

The Cassandra data model was initially proposed by Google, Inc. under the name Bigtable [58]. In 2006, Google, Inc. published the Bigtable concept and described its use in powering various Google services. Despite giving a clear description on the data model, Google did not release its Bigtable software to the public.

Lacking a publicly available implementation of the Bigtable concept, Cassandra development was started at Facebook, Inc. One of the core developers, Avinash Lakshman, had previously worked for Amazon's DynamoDB NoSQL database. Probably for this reason, Cassandra also provides features originally found in DynamoDB such as the data replication model.

In 2008, Facebook, Inc. released Cassandra as open-source software. Shortly after the release, Cassandra got accepted into the Apache Software Foundation's incubator program. In 2010, Cassandra graduated into the list of top-level projects of the Apache Software Foundation.

Another attempt of reimplementing the Bigtable concept is made in the HBase [59] project. HBase is also part of the Apache software foundation and has a stronger focus on Bigtable compatibility than Cassandra. Meanwhile, Google has also made their own Bigtable implementation available to customers of their cloud services with an HBase compliant API.

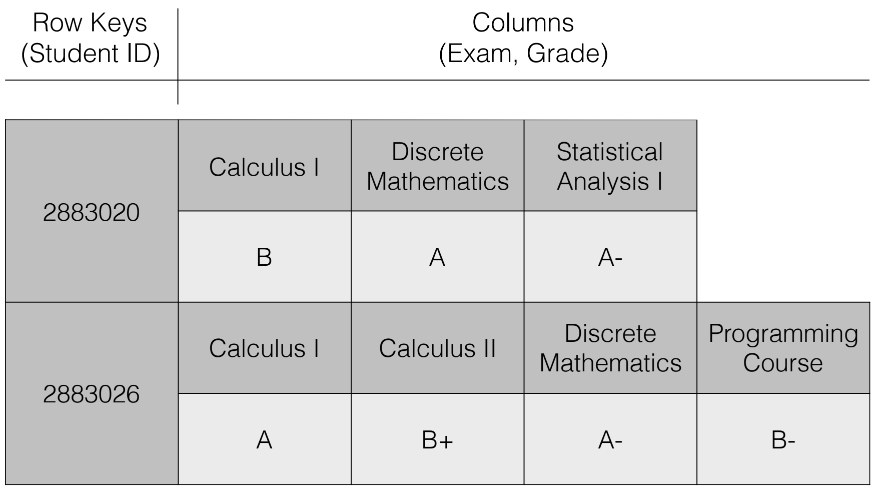
4.2.2 *Data Model*

In comparison to traditional relational database management systems, the Cassandra data model is very restrictive. For example, Cassandra does not support joins on different data sources. In return, all operations within Cassandra are inherently scalable and performance increases nearly linearly with the number of database nodes.

The fundamental concept of Cassandra's data model is to store data in rows with a unique row key. All data operations on the database require the row key to be specified. For each row, Cassandra keeps a list of columns consisting of a column name and value. Columns are automatically sorted in ascending order according to their name which enables Cassandra to perform range queries on column names.

Similar to the concept of tables in a relational database, multiple rows and their columns are officially called a "column family" in Cassandra terms. Yet, the term "table" is used synonymously with the term "column family" in Cassandra's syntax and documentation. For separating different applications, Cassandra supports grouping multiple column families in keyspaces.

In addition to column name and value, each column also contains a timestamp of when the data was written and, if needed, also a time to live entry for automatically retiring data after a predefined period.



Row Keys (Student ID)	Columns (Exam, Grade)			
2883020	Calculus I	Discrete Mathematics	Statistical Analysis I	
	B	A	A-	
2883026	Calculus I	Calculus II	Discrete Mathematics	Programming Course
	A	B+	A-	B-

Figure 4: Example of a Cassandra column family storing the examination results of students.

As opposed to the table model in relational databases, column names do not need to be identical across rows.

Figure 4 provides an example of modeling the examination results of students as a Cassandra column family. The unique student IDs are used as row keys, the course names act as column names. Column values contain the grades. It shows that column names do not need to be identical across rows as well as the automatic ordering of columns by name.

Since all data operations require the row key to be specified, data can be split according to their row key onto different database servers. Using a hash function that evaluates in constant time, each row key can be mapped onto a binary value of 128 bits. This hash function for row keys is called a partitioner since it defines the partitioning of rows onto multiple database servers. Subsequently, the term "partition key" is used synonymously with the term "row key". For the partitioning, each server is assigned a certain interval in the 128 bit value space of the hash function. In the default configuration of Cassandra, a partitioner is used that results in a relatively random distribution for various types of data. This setup is meant for cases where a set of database servers shares the total load and the partitioning provides increased storage space and throughput. Alternatively, Cassandra implements a byte-ordered partitioner that does not implement a hash function, thus, mapping unsigned integer values of row keys directly onto the configured database server partitions. Only with the byte-ordered partitioner, Cassandra supports range queries on row keys in addition to range queries on column names. In return, use of the byte-ordered partitioner requires Cassandra users to take special care of the load balancing through data distribution within their application logic.

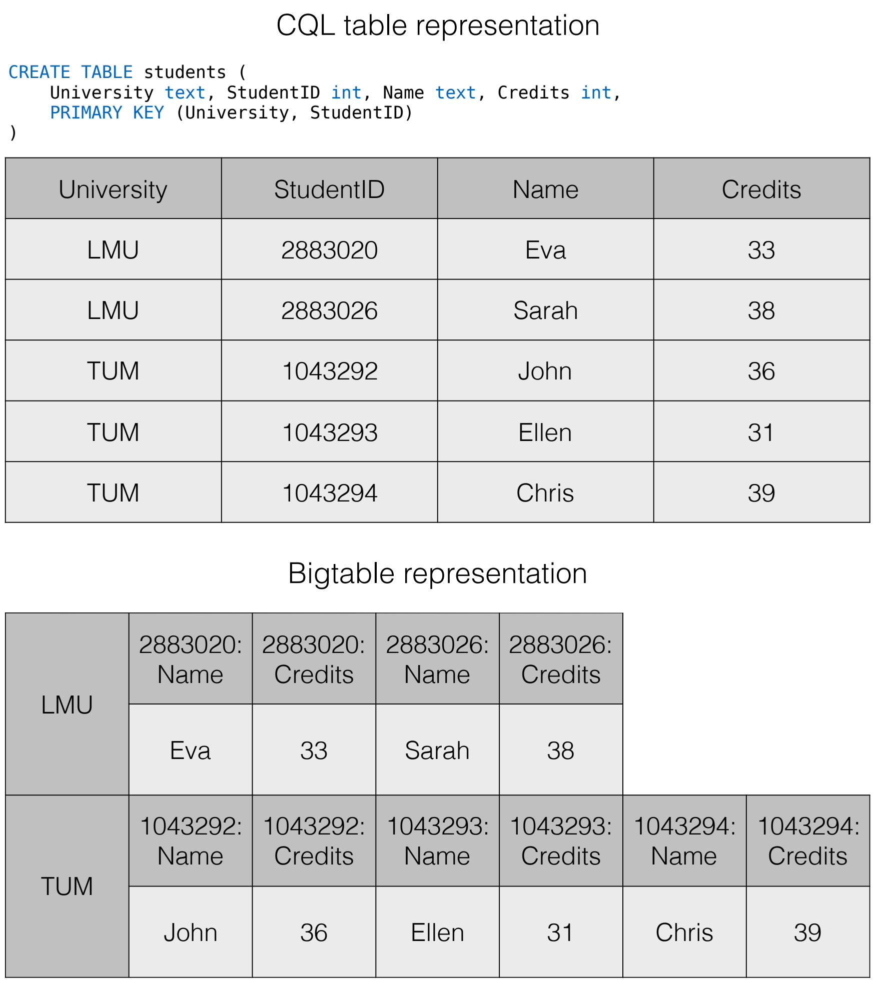
Besides distributing data over multiple servers, Cassandra also provides means for data replication. Replication yields increased performance in a distributed setup since multiple servers can serve the same data in parallel. In addition, replication is an obvious approach for increased data availability. Cassandra allows the configuration of a replication factor for each keyspace and then automatically handles the copying of data onto multiple nodes. Since replication does not happen synchronously, conflicting inserts into the database can be resolved by comparing the time stamps associated with each column. For retrieving data, users can configure a quorum that has to be met from multiple database nodes before column data is returned to the user.

4.2.3 *User and Application Programming Interfaces*

For interfacing with external applications, Cassandra provides multiple interfaces. Until Cassandra version 0.8, the only method for interacting with Cassandra were remote procedure calls (RPC) over a Thrift-generated interface. Apache Thrift [60] is a software that generates RPC interface bindings for different programming languages from a common interface definition file. Through the use of Thrift, Cassandra databases could be used in applications written in C++, Java, Python, PHP, and many others. The Thrift API for Cassandra was based on the direct manipulation of low level database objects. As such, programmers had to deal with implementation details such as the internal timestamps that Cassandra uses to resolve concurrent conflicting inserts into the database.

To overcome the complexity of the Thrift API, Cassandra 0.8 introduced the Cassandra Query Language (CQL) [61]. CQL has similarities to the Structured Query Language (SQL) used by traditional relational database management systems. However, CQL only provides a subset of the SQL syntax, restricting the syntax to functionalities that are available within the limits of the underlying Cassandra data model. Still, CQL facilitates the practice of mapping SQL-like table schemas onto the schema-less Bigtable data model using compound row and column names as illustrated in Figure 5. Along with the introduction of CQL, the Cassandra developers added a new networking protocol for interfacing with Cassandra databases using the CQL language. Due to the open-source nature of Cassandra, bindings for many programming languages (C++, Java, Python, PHP...) have been developed that expose the new CQL-based interface to application programmers. Following contiguous improvements of CQL, the Thrift interface has been removed in Cassandra version 3.

Besides the application programming interfaces, Cassandra ships with a set of command line tools for administration and data manipulation. The `cqlsh` tool provides a CQL shell for the immediate



CQL table representation

```
CREATE TABLE students (
    University text, StudentID int, Name text, Credits int,
    PRIMARY KEY (University, StudentID)
)
```

University	StudentID	Name	Credits
LMU	2883020	Eva	33
LMU	2883026	Sarah	38
TUM	1043292	John	36
TUM	1043293	Ellen	31
TUM	1043294	Chris	39

Bigtable representation

LMU	2883020: Name	2883020: Credits	2883026: Name	2883026: Credits		
	Eva	33	Sarah	38		
TUM	1043292: Name	1043292: Credits	1043293: Name	1043293: Credits	1043294: Name	1043294: Credits
	John	36	Ellen	31	Chris	39

Figure 5: Example of the CQL approach on mapping SQL-like tables to the Bigtable data model.

execution of CQL queries. The `nodetool` command provides administrative functions for Cassandra clusters such as database repairs and load monitoring.

4.3 STORING TIME SERIES OF SENSOR DATA

Cassandra provides extreme scalability at the cost of a significantly reduced data model. Thus, the concept of using Cassandra for storing sensor data promises to be successful if a method is found to describe the problem of handling time series of sensor data within the Cassandra data model. The following section describes such a method and the presumptions leading to it.

4.3.1 *Mapping Time Series to the Cassandra Data Model*

For the storing of time series containing sensor data, one can make the following assumptions on the usage pattern:

- Each sensor has a unique identifier S.
- Each entry in the database is a triplet (S, T, V) with T being the time of the reading, and V being the sensor's value at time T.
- A full (S, T, V) triplet is provided for every valid insert (i.e. there are no inserts consisting just of S and T without V).
- For a given sensor S at a given time T, there must be no more than one value V.
- Queries into the database have the form $(S, [T_0, T_1]) \to (T, V)^*$ with S being the sensor's identifier and $[T_0, T_1]$ being the time interval of interest between start time T_0 and end time T_1. The query result $(T, V)^*$ is a list of timestamp-value pairs (T, V) for the sensor S in the requested time interval.

The assumption that sensor identifiers S are unique fulfills a necessary constraint for using S as row keys in Cassandra. Additionally, according to the usage pattern described above, S is always known for both, inserts and queries. This is a sufficient requirement for using S as row keys. With the intrinsic ordering of columns in Cassandra and the resulting support for range queries on column names, the column name fields provide the best location to store timestamps T. This leaves the column's value field for the sensor's value V.

To achieve this mapping of S to row keys, T to column names, and V to column values, DCDB uses the following CQL statement for creating the sensor data table:

Listing 1: Creation of the sensor data table with CQL.

```
CREATE TABLE sensordata (sid BLOB, ts BIGINT, value BIGINT, PRIMARY KEY (
    sid, ts)) WITH COMPACT STORAGE;
```

This CQL statement creates a new table named `sensordata` with three fields:

- The sensor identifier S (`sid`)
- The time stamp T (`ts`)
- The sensor value V (`value`)

Through the `PRIMARY KEY (sid, ts)` statement, Cassandra will automatically use the values of the `sid` field as row keys and the values of the `ts` field as column names.

The `COMPACT STORAGE` directive in the statement is an optimization flag that reduces the disk space required for the table. In return, CQL tables created with `COMPACT STORAGE` directive can only contain one field besides the primary key fields and altering the table definition after its creation is not allowed.

4.3.2 *Data Types and Storage Conventions*

The `CREATE TABLE` statement from Listing 1 also defines the data types used for storing sensor data within DCDB.

In DCDB, the sensor identifier S stored in the `sid` field is represented as a binary structure of 128 bits. Unfortunately, Cassandra only provides data types for storing integer numbers of up to 64 bits. It does, however, provide the `BLOB` data type that represents a container for binary data of arbitrary size. Thus, in DCDB, the `sid` field uses the `BLOB` data type and DCDB's application logic ensures that all values are exactly 128 bits wide. In conjunction with the byte-ordered partitioner, the value in this field then also enables control over the storage location of a sensor's data row.

The time stamp T stored in the `ts` field is defined as data of type `BIGINT`. The `BIGINT` data type in Cassandra holds signed integer values with 64 bits precision. In computing, processing time information using integer values is common practice as it allows for faster operations on time data than processing time represented as character strings. For example, UNIX timestamps are defined as integer values representing the elapsed time in seconds since the UNIX epoch (January 1, 1970). To facilitate interfacing with other applications using UNIX timestamps while, at the same time, increasing the temporal resolution, DCDB uses nanoseconds since UNIX epoch within the `ts` database field. Similar to UNIX timestamps, DCDB timestamps represent time in UTC. This ensures that the DCDB timestamp function is monotonic rising which avoids possible issues related to time zones and daylight savings time.

DCDB also uses the `BIGINT` data type for representing the sensor values V in the `value` field. This is a debatable choice, since sensor data often originates from analog sensors measuring temperatures, voltages, etc. for which floating point data might seem more appropriate. However, in current computing equipment, the conversion from the analog measurement to a digital value is done in tiny microcontrollers that do not support the processing of floating point values in hardware. To work around this hardware limitation, microcontroller developers could implement floating point support in software which comes with a significant performance penalty. Therefore, the use of integers representing multiples of the actual base unit (e.g. millivolts instead of volts, tenths of °C instead of °C, . . .) is used instead.

With the goal of collecting data directly from as many sources as possible and with respect to the missing floating point capabilities of most microcontrollers, DCDB uses integers for storing sensor data. Since this yields to DCDB storing data in a variety of different units, DCDB supports transparent conversion of units when reading data from the database. The DCDB unit conversion framework will be explained in Section 4.7.3.

4.3.3 *Defining Sensor Identifiers*

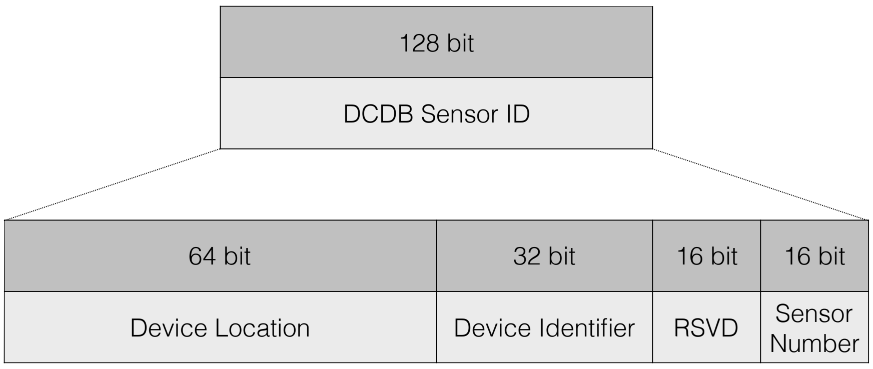


Figure 6: Structure of DCDB sensor IDs.

As explained in the previous section, DCDB sensor identifiers are 128 bits wide to match with Cassandra's internal representation for row keys. DCDB assumes that sensors are contained in devices (e.g. compute nodes, network switches...). It also assumes that devices are characterized by their physical location and a unique device identifier that persists even when the device is relocated to a different physical position.

Figure 6 shows, how DCDB sensor identifiers are composed to reflect this model:

DEVICE LOCATION describes the (unique) physical location of the device. Users of DCDB are free to assign their own scheme of encoding a physical location into this 64 bit wide field. However, since this field makes up for the most significant bits of the sensor identifier, it directly influences how Cassandra will split sensor data onto different database nodes. Users must therefore be careful in modeling device locations in a way that allows mapping this data to nearby Cassandra database nodes.

DEVICE IDENTIFIER holds an identifier that is unique to a device which is independent from its position, i.e. it persists even when the device is moved to a different location. For example, device identifiers can be derived from a device's MAC address or serial number. Being able to track devices independent from their physical location is key to an efficient analysis of data center operations: for example, when tracking down issues of insufficient cooling, it is important to know whether the problem is within the device (i.e. broken fan) or from the device's location (i.e. hot spot due to turbulences in the room's cooling air distribution).

RSVD is a reserved field. It will be described in the next paragraph.

SENSOR NUMBER is an identifier of the sensor within a device. This number can be assigned arbitrarily by DCDB users, although it is recommendable to use the same sensor numbering schemes across similar devices.

4.3.4 *Coping with Cassandra's Wide Row Limits*

Cassandra encourages the use of wide rows, that is rows with many pairs of column name and value per row key. However, Cassandra also carries a design limitation that the number of cells per row may not exceed 2 billion. When running DCDB for a long time or when incorporating sensors with a high read-out frequency, this limit could be reached. In order to work around this limitation of Cassandra, DCDB transparently inserts a time based hash into the sensor ID. This hash, also referred to as *week stamp* is calculated for every insert as the number of weeks since the UNIX epoch for the given timestamp T. The week stamp is then imprinted into the sensor ID's 16-bit wide `RSVD` field. With this concept, the identifier S stored in the `sid` field for a given sensor will change every week. Thus, the Cassandra limitation of cells per row no longer imposes a global limit on the number of data points per sensor. Instead, DCDB's limitation is now 2 billion data points per week per sensor, which is reasonably large.

Calculating the week stamp transparently upon every insertion of a data point only consists of one division and the bit arithmetic to insert the week stamp into the sensor ID. More work has to be done, however, for honoring the week stamp during queries into the sensor data store since a query may span multiple weeks and the sensor's time series has to be assembled from multiple rows within Cassandra. However, DCDB has all the necessary functionality implemented to handle inserts and queries without the need for DCDB users to be aware of the week stamp concept.

4.4 MQ TELEMETRY TRANSPORT

To collect sensor data from various sources, DCDB relies on the MQ Telemetry Transport (MQTT) messaging protocol [62]. MQTT was invented in 1999 and originated from a collaboration between IBM and Eurotech. In 2015, MQTT has been adopted as an OASIS standard

[63] and in 2016, MQTT has been adopted by the ISO/IEC technical committee on information technology[2].

The MQTT protocol is designed to be lightweight and simple to use. Also, the small size of the reference implementation allows for using it even on tiny microcontrollers.

In its original form, MQTT uses a publish-subscribe model in which MQTT *clients* connect to a *broker*. A client then either publishes messages to the broker or subscribes to the broker with a list of topics of interest. It is then the broker's responsibility to forward all messages that it receives according to the clients' subscriptions. In the current implementation of DCDB, however, the intermediate use of a broker and explicit subscriptions to topics does not take place. Instead, any MQTT message sent into DCDB will result in an insertion of the sensor data contained within the message into the database. Yet, by sticking to the MQTT message format, DCDB can be extended to interact with other MQTT-capable tools without problems in the future.

Each MQTT message consists of a message topic and a message body. Within the DCDB framework, the message topic is a character string, representing the 128 bit sensor ID in hexadecimal notation. For increased readability (e.g. to separate the device location, device identifier, and sensor number fields), the number may be split using forward-slash characters. The RSVD field in which DCDB will place the week stamp can contain arbitrary data as it will be overwritten by DCDB.

For the message body, DCDB supports two different formats, depending on whether the device sending the message has the capability to provide time stamp information or not. In case the message originates from a data source that has no access to accurate real time clock information, the message body solely consists of a 64 bit number representing the sensor's current reading. Upon receiving the message, DCDB will assign the current time to the reading and insert it into the database. If the message originated from a data source that has the ability to provide real time clock information, DCDB also accepts a list of timestamp-value pairs within the message body. The list of values will then be inserted into the database with the provided timestamps. The latter message format also allows for transferring multiple data points within the same message, thus reducing the protocol overhead per data point and increasing the data insertion bandwidth.

4.5 COLLECT AGENT

The previous sections explained how DCDB stores time series data using the Cassandra database and how the MQTT protocol is used to transmit sensor data. This section introduces *Collect Agent*, which

2 ISO/IEC 20922:2016

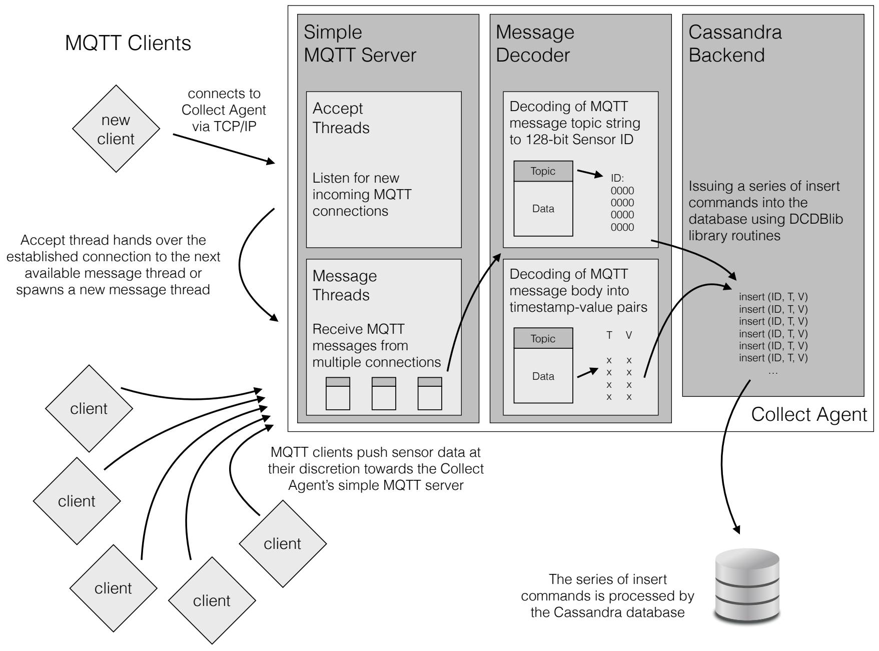

MQTT Clients
new client
connects to Collect Agent via TCP/IP
Accept thread hands over the established connection to the next available message thread or spawns a new message thread
client
client
client
client
client
MQTT clients push sensor data at their discretion towards the Collect Agent's simple MQTT server
Simple MQTT Server
Accept Threads
Listen for new incoming MQTT connections
Message Threads
Receive MQTT messages from multiple connections
Message Decoder
Decoding of MQTT message topic string to 128-bit Sensor ID
Topic
Data
ID: 0000 0000 0000 0000
Decoding of MQTT message body into timestamp-value pairs
T V
Cassandra Backend
Issuing a series of insert commands into the database using DCDBlib library routines
insert (ID, T, V)
Collect Agent
The series of insert commands is processed by the Cassandra database

Figure 7: Graphical overview of the Collect Agent tasks and data flow

is the software component of DCDB that receives sensor data from MQTT messages and writes it into the Cassandra database.

Collect Agent is a C++ program designed to run as a server daemon. During the startup, Collect Agent uses the DCDB API to connect to the Cassandra database. Once connected, it ensures that all keyspaces and column families are present and creates them, if necessary. It then starts its internal MQTT server. The MQTT server of Collect Agent implements a subset of MQTT commands sufficient to receive MQTT messages with sensor data. Upon receiving of a MQTT message, Collect Agent tries to decode the message topic string into a 128 bit sensor ID by stripping any forward-slash characters. Once successful, Collect Agent checks the message payload to determine whether the message consist of just a value, a timestamp value pair, or a series of timestamp value pairs. It then uses its existing connection to the Cassandra database to insert the data into the `sensordata` column family. Figure 7 visualizes the Collect Agent's components and the associated data flow.

For increased performance, Collect Agent implements a threaded design. Each socket on which Collect Agent listens for incoming connections runs in its own thread. Once a client connects to the MQTT server, the connection is being handed over to one of multiple connection handling threads. In oder to optimize thread usage, each connection handling thread is capable of working multiple concurrent connections. Both Collect Agent and the underlying libraries for accessing Cassandra are implemented in a thread-safe fashion so that the insertion of data up until the actual database operation can be performed in parallel.

Large DCDB installations will run multiple Cassandra database nodes as well as multiple instances of the Collect Agent. A reasonable approach is to run Collect Agent and its associated Cassandra service on the same server. However, for increased performance, the two can also be run on separate servers.

4.6 SOURCES FOR SENSOR DATA

Recent years have seen a contiguous decline in cost, size, and power consumption in sensor technology and associated measurement electronics. This trend has enabled market growth for cyber-physical systems and is accompanied with marketing labels such as the *Internet of Things (IoT)* and *Smart Sensors*. DCDB was designed for such environments and the deliberate choice for MQTT as the protocol for transferring sensor data was made due to the popularity of MQTT in the IoT scene and its generally wide support.

In an ideal DCDB monitoring setup, one would rely solely on smart sensors that contain a microcontroller capable of establishing IP-based connections to the Collect Agent. The sensor would then be assigned a Sensor ID and, once it is powered up and connected to the network, push its information autonomously towards the Collect Agent.

On today's high performance computers, however, any monitoring system has to work with existing protocols and data paths for the acquisition of sensor data. Widely used protocols are IPMI and SNMP for accessing out-of-band data. As an example to demonstrate the amount of information from a single compute node, Table 4 lists all sensors on a current popular HPC platform that are available out-of-band via IPMI. In-band data is often available from the Linux sysfs and proc pseudo file systems.

To cope with these existing infrastructures, a set of *Pushers* has been developed. The Pushers are software components within DCDB, which interface between the existing methods for data acquisition and the DCDB scheme of pushing data via MQTT messages.

4.6.1 *IPMI Pusher*

The Intelligent Platform Management Interface (IPMI) is an industry standard for server management [64]. The IPMI standard is administered by Intel®. Meanwhile, more than 200 companies have adopted or contributed to the IPMI standard or are registered otherwise as industry promoters [65].

The documents describing the IPMI standard are publicly available. They describe the hardware configuration and internal commu-

Sensor	*Value*	*Sensor*	*Value*	*Sensor*	*Value*
BB Inlet Temp	27 degrees C	P1 Therm Margin	-58 degrees C	System Event Log	0x00
BB BMC Temp	33 degrees C	P2 Therm Margin	-61 degrees C	System Event	0x00
BB CPU1 VR Temp	34 degrees C	P1 Therm Ctrl %	0 percent	Button	0x00
BB CPU2 VR Temp	32 degrees C	P2 Therm Ctrl %	0 percent	BMC Watchdog	0x00
BB MISC VR Temp	36 degrees C	P1 DTS Therm Mgn	-48 degrees C	VR Watchdog	0x00
BB Outlet Temp	34 degrees C	P2 DTS Therm Mgn	-51 degrees C	SSB Therm Trip	0x00
System Airflow	24 CFM	P1 DIMM Thrm 1	-42 degrees C	BMC FW Health	0x00
LSI3008 Temp	29 degrees C	P1 DIMM Thrm 2	-53 degrees C	NM Capabilities	0x00
SSB Temp	35 degrees C	P2 DIMM Thrm 1	-44 degrees C	Auto Shutdown	0x00
HSBP PSOC	27 degrees C	P2 DIMM Thrm 2	-54 degrees C	PS1 Status	0x00
Exit Air Temp	34 degrees C	Agg Therm Mrgn 1	-26 degrees C	PS2 Status	0x00
LAN NIC Temp	38 degrees C	Agg Therm Mrgn 2	-33 degrees C	P1 Status	0x00
Sys Fan 1A	13410 RPM	HSBP Temp	24 degrees C	P2 Status	0x00
Sys Fan 1B	13950 RPM	Riser 1 Temp	29 degrees C	CPU ERR2	0x00
Sys Fan 2A	13320 RPM	Riser 2 Temp	26 degrees C	CPU Missing	0x00
Sys Fan 2B	13857 RPM	BB +12.0V	12.10 Volts	VRD Hot	0x00
MTT CPU1	0 percent	BB +3.3V Vbat	3.02 Volts	PS1 Fan1 Fail	0x00
Sys Fan 3A	12870 RPM	Mem 1 VRD Temp	32 degrees C	PS1 Fan2 Fail	0x00
MTT CPU2	0 percent	Mem 2 VRD Temp	27 degrees C	PS2 Fan1 Fail	0x00
Sys Fan 3B	14043 RPM	EV CPU1VR Temp	31 degrees C	PS2 Fan2 Fail	0x00
PS1 Input Power	299 Watts	Pwr Unit Status	0x00	Mem P1 Thrm Trip	0x00
PS2 Input Power	13 Watts	Pwr Unit Redund	0x00	Mem P2 Thrm Trip	0x00
PS1 Curr Out %	10 percent	IPMI Watchdog	0x00	Voltage Fault	0x00
PS2 Curr Out %	0 percent	Physical Scrty	0x00	HDD 0 Status	0x00
PS1 Temperature	27 degrees C	FP NMI Diag Int	0x00	HDD 1 Status	0x00
PS2 Temperature	27 degrees C	SMI Timeout	0x00	HDD 2 Status	0x00

Table 4: Out-of-band sensors on an Intel® S2600BP mainboard

nication paths of IPMI compliant computer systems as well as the interface and protocol for out-of-band communication with outside monitoring and management systems.

The central element in any implementation of the IPMI standard is the baseboard management controller (BMC). The BMC is a system-on-chip device running a full multi-tasking operating system. It is connected to many of the server's internal system management busses and emulates common input/output devices such as a video screen and keyboard to forward screen contents and keyboard input between the server and remote management clients. Using this, system administrators can remotely see screen content and send commands to the server already during the system's early initialization phases, even before regular networking to the server has been configured.

Multiple software development kits for developing IPMI firmware for BMCs are available and various open-source software libraries exist to interact with IPMI compliant BMCs over a system-internal connection as well as remotely over a serial connection or LAN. IPMI network connections make use of the user datagram protocol (UDP), and thus require additional error handling means in the application logic.

For the DCDB framework and its monitoring purposes, only the IPMI subsystem that provides access to system sensors is relevant. IPMI Pusher is a C++ application that accesses the IPMI sensors using the routines provided by the OpenIPMI library [66]. It features the ability to connect to multiple BMCs at once using a threaded design with one thread per connection. A configuration file contains the hostnames and login credentials of all BMCs. It also contains a list of sensors to be queried as well as the desired sensor readout frequency. A central thread maintains a priority queue that contains all sensors to be queried as well as the time of the next reading. Once the next reading is due, the due entry is taken from the priority queue and the thread handling the respective BMC is notified to initiate a read sensor request. After this, the time of the next reading is calculated based on the configured readout interval of the sensor and the sensor is re-inserted into the priority queue.

Since BMC requests are performed asynchronously, response messages from the BMCs can arrive at any time. Whenever a successful sensor reading arrives at one of the BMC connection management threads, the thread assigns a timestamp to the reading and initiates the creation of a MQTT message containing the sensor ID and the value. The MQTT message is then sent to the Collect Agent.

4.6.2 *SNMP Pusher*

The Simple Network Management Protocol (SNMP) is an industry standard communication protocol for managing IT devices over a network. In contrast to IPMI, which is designed explicitly for remote server management, SNMP features a simpler and more generic design. With the exception of special SNMP trap messages, SNMP implements a simple master-slave communications approach. In the SNMP terminology, the master (i.e. a management server) is called *manager* whereas the slaves (i.e. a SNMP capable network switch) are called *agents*.

The manager communicates to the agents using *get* and *set* commands to read or change values in the device. Values accessible via SNMP can be configuration options, sensor readings, or general information such as the vendor, device type and firmware version of a device. To access a given value, the manager needs to specify the value's object identifier (OID) in the request. For this, SNMP relies on a global unique OID space as established jointly by the International Telecommunication Union (ITU) and the International Organization for Standardization's International Engineering Consortium (ISO/IEC).

The interest in supporting SNMP within the DCDB framework is twofold: first, many IT components (such as network switches, routers, or storage appliances) provide built-in support for monitoring via SNMP. Secondly, a variety of devices exist to bridge other, less-common protocols to SNMP. An example for such a bridge device is explained in Chapter 5, where the DCDB based monitoring system collects data from the data center's cooling system controller.

Similar to IPMI Pusher, SNMP Pusher is a C++ application that acts as SNMP manager to query SNMP agents and forwards the collected information via MQTT messages into DCDB. To access SNMP agents, SNMP Pusher relies on the functions provided by the Net-SNMP library. The Net-SNMP software suite is BSD-licensed and contains a library and accompanying tools implementing the SNMP protocol [67]. Similar to IPMI, SNMP communicates via UDP and additional efforts have to be spent in the application logic to deal with the handling of packet errors.

The rest of SNMP Pusher works identically to IPMI Pusher with a priority queue to determine the next sensor and OID to be queried via SNMP.

4.6.3 *sysfs Pusher*

The Linux *sysfs* pseudo file system is one of multiple interfaces between the Linux Kernel along with its device drivers and user space applications. In modern Linux distributions, sysfs provides a directory structure under the `/sys` mount point. Obtaining information and changing settings is performed by reading from or writing to files in this directory structure.

An example for a driver that is accessible via sysfs is the *cpufreq* driver. Intel® processors since the Pentium M series can adjust their processor frequency during runtime to reduce the processor's speed and power consumption during periods in which the computational load is low. Other processor designs, in particular in the embedded field provide similar features. The cpufreq driver interacts with the processor's clock generation subsystem to control the behavior of the CPU-internal clock selection mechanism. For this, cpufreq accepts values for minimum and maximum processor frequencies through sysfs and allows for selecting a so called *governor* which controls how the frequency is adapted based on the current workload. For system monitoring, the cpufreq sysfs interface also provides information about the actual frequency as selected by either the governor or the processor itself.

In the context of DCDB and system monitoring for energy efficiency, it is also noteworthy that system integrators such as IBM, Lenovo, Cray, and Megware provide additional hardware for fine-grained energy monitoring. All mentioned companies provide Linux drivers that make the values of the energy counters available through sysfs.

DCDB's sysfs Pusher follows the ideas of the IPMI and SNMP Pushers by forwarding data obtained through the sysfs pseudo file system to Collect Agent via MQTT messages. The sysfs Pusher is a C++ application that runs as a daemon on each compute node. In regular, configurable intervals, it reads a set of configured values from sysfs, packs them as MQTT message and sends it to the Collect Agent.

4.6.4 *File Pusher*

As the name suggests, File Pusher is a program that reads the contents of arbitrary files, and after interpretation, forwards the file contents to the Collect Agent via MQTT. Its main purpose is to serve as test application for the development of DCDB. It behaves almost identical to the sysfs Pusher which also obtains its data by simply issuing read commands on local files. The only extra feature of File Pusher over the sysfs Pusher is the ability to use the Linux inotify API to generate a new MQTT message whenever the contents of a file change. Since the inotify API does not work on pseudo file systems such as sysfs, this feature is missing from sysfs Pusher.

4.7 SENSOR MANAGEMENT

The previous sections of this chapter have covered the means by which DCDB collects, forwards, and stores sensor data for system monitoring. When rolling out a DCDB installation for a high performance computing system, careful planning is required that ensures a well-performing setup:

- Which devices shall be included in the monitoring setup?
- What are the best locations in the network topology to run the Collect Agents and Cassandra servers?
- What is the scheme for assigning sensor IDs?

Unfortunately, manual planning of the DCDB setup around these questions is unavoidable since every system is different and the features of DCDB's distributed data acquisition and storage approach are only exploitable when knowledge of the system and knowledge about DCDB's internal structures are well aligned. However, once the data acquisition infrastructure is in place, DCDB makes accessing the stored data comfortable and easy.

4.7.1 *Publishing Sensors*

For performance reasons, DCDB stores all incoming sensor data in raw 64-bit integer values in the sensordata column family. In particular, DCDB stores no meta information except for the sensor ID and the timestamp along with the raw sensor data. Within DCDB, the time series of raw data are treated as internal data without the need for direct user access.

In order to make data accessible, DCDB provides the concept of *publishing* sensors. The publishing of a sensor is done for two reasons. First, a published sensor carries additional meta data such as a freely assignable name and the physical unit that the raw values represent. Second, under the assigned public name, the sensor does not only represent a single internal 128-bit sensor ID. Instead, it is associated with a pattern which is matched against all internal sensor ids. As explained in Section 4.3.3, the internal 128-bit sensor ID is composed of a device location, a device identifier, and a sensor number. DCDB assumes that, in most cases, users would want to query sensor data either by the device's location or by the device's unique identifier. For example, a user would like to know the CPU temperature of the top most compute node in a given rack. In this case, the device identifier (e.g. serial number of the compute node) would not matter and the query should return the requested data independent from the actual device operating in the location.

In return, when trying to track a faulty device across different locations, a user would want to access sensor data originating from the device independent from the location of the device at a given time. Due to the ability of DCDB to match against multiple internal 128-bit sensor IDs, queries of the aforementioned type become possible.

In practice, when creating a public sensor, the user specifies a pattern which consists of a raw sensor ID and allows for a wildcard (*) character. Whenever a public sensor's purpose is to represent a sensor from a device at a given location, the device ID field would be replaced with the wildcard. Similarly, when a public sensor's purpose is to represent a sensor from a specific device independent from the device's location, the device location part of the sensor ID would be replaced with the wildcard.

For the publishing of sensors, DCDB provides a command line utility: `dcdbconfig`. Internally, DCDB maintains a column family called `publishedsensors` and any invocation of the dcdbconfig utility will operate on this column family.

Listing 2: CQL statement for creating the configuration table which holds all published sensors.

```
CREATE TABLE publishedsensors (
    name VARCHAR,            /* Public name */
    virtual BOOLEAN,         /* Whether it is a published physical sensor
                                or a virtual sensor */
    pattern VARCHAR,         /* For physical sensors: pattern for SIDs
                               that this will match */
    scaling_factor DOUBLE,   /* Optional scaling factor that will be
                               applied upon readout */
    unit VARCHAR,            /* Unit of the sensor (e.g. W for Watts) */
    integrable BOOLEAN,      /* Indicates whether the sensor is integrable
                                over time */
    expression VARCHAR,      /* For virtual sensors: arithmetic expression
                                for the sensor */
    vsensorid VARCHAR,       /* For virtual sensors: sensorId in the
                               virtualsensors table */
    tzero BIGINT,            /* For virtual sensors: time of the first
                               reading */
    frequency BIGINT,        /* For virtual sensors: readout frequency for
                                the virtual sensor */
    PRIMARY KEY(name))
    WITH COMPACT STORAGE AND CACHING = all;  /* Enable compact storage
                               and maximum caching */
```

While the `name` and `pattern` fields have been subject to this section, the additional fields in this column family will be explained in the following sections.

4.7.2 *The DCDB Query Tool*

Publishing a sensor by assigning a name and sensor ID pattern is the minimum prerequisite for receiving data stored within DCDB. Once published, time series of sensor data can be retrieved with the `dcdbquery` command line utility. In its simplest invocation, `dcdbquery` requires a public sensor name, a value for the start time and a value for the end time of the time series. `dcdbquery` will then use the DCDB API to connect to Cassandra, retrieve the data and output the data as comma separated value (CSV) list.

For convenience, `dcdbquery` supports querying multiple sensors at once. In addition, `dcdbquery` provides flexible means to specify the start and end time for the query. Supported time formats are:

- Absolute time according to ISO 8601 [68]
  Example: `2017-01-31 23:59:59.000`

- Absolute time using POSIX time
  POSIX time counts the number of seconds elapsed since the Unix epoch (1970-01-01 00:00:00).
  For increased granularity, DCDB also accepts milliseconds, microseconds and nanoseconds since the Unix epoch.
  Example: `1485907199`

- Relative time to current time
  Using the keyword `now`, users can specify any point in time prior to the current time by specifying an offset of days (`d`), hours (`h`), minutes (`m`), or seconds (`s`).
  Example: `now-1h`

To retrieve the time series data from Cassandra, DCDB performs multiple steps. First, the specified sensor name is looked up in the publishedsensors table to get the sensor ID pattern string. Due to DCDB's Cassandra database schema, queries into the database have to specify an exact 128-bit sensor ID. While a single 128 bit sensor ID constitutes a valid pattern, the pattern might also contain a wildcard character. Hence, patterns containing a wildcard need to be expanded into a matching list of existing sensor IDs first. As explained in Section 4.3.4, the sensor IDs also contain a week stamp to cope with Cassandra's wide row limits. Therefore, the list of sensor IDs needs to be further expanded to contain one sensor ID per weekstamp per physical sensor. Finally, with the resulting set of sensor IDs, DCDB can query the sensor data for each sensor ID in the set in the given range to generate the resulting data set. Due to the ordering of columns in Cassandra, the result will automatically be ordered ascending by timestamp.

4.7.3 *DCDB Unit Conversion Framework*

A benefit of the concept of publishing sensors lies in the ability to augment a public sensor with additional information such as the physical unit that the raw data represents. DCDB has a basic understanding of physical units and is able to perform linear conversions between units. For this, DCDB maintains a table of physical units.

Listing 3: Definition of the DCDB unit conversion table.

```
ConversionTableEntry conversionTable[] = {
/*          Unit Name          Symbol  Base Unit      Scaling  Offset */
/*  0 */  { Unit_None,         "none", Unit_None,          1, 0 },
/*  1 */  { Unit_Meter,        "m",    Unit_Meter,         1, 0 },
/*  2 */  { Unit_CentiMeter,   "cm",   Unit_Meter,      -100, 0 },
/*  3 */  { Unit_MilliMeter,   "mm",   Unit_Meter,     -1000, 0 },
/*  4 */  { Unit_MicroMeter,   "um",   Unit_Meter,  -1000000, 0 },
/*  5 */  { Unit_Second,       "s",    Unit_Second,        1, 0 },
/*  6 */  { Unit_MilliSecond,  "ms",   Unit_Second,    -1000, 0 },
/*  7 */  { Unit_MicroSecond,  "us",   Unit_Second, -1000000, 0 },
/*  8 */  { Unit_Ampere,       "A",    Unit_Ampere,        1, 0 },
/*  9 */  { Unit_MilliAmpere,  "mA",   Unit_Ampere,    -1000, 0 },
/* 10 */  { Unit_MicroAmpere,  "uA",   Unit_Ampere, -1000000, 0 },
/* 11 */  { Unit_Kelvin,       "K",    Unit_Kelvin,        1, 0 },
/* 12 */  { Unit_MilliKelvin,  "mK",   Unit_Kelvin,    -1000, 0 },
/* 13 */  { Unit_MicroKelvin,  "uK",   Unit_Kelvin, -1000000, 0 },
/* 14 */  { Unit_Watt,         "W",    Unit_Watt,          1, 0 },
/* 15 */  { Unit_MilliWatt,    "mW",   Unit_Watt,      -1000, 0 },
/* 16 */  { Unit_MicroWatt,    "uW",   Unit_Watt,   -1000000, 0 },
/* 17 */  { Unit_KiloWatt,     "kW",   Unit_Watt,       1000, 0 },
/* 18 */  { Unit_MegaWatt,     "MW",   Unit_Watt,    1000000, 0 },
/* 19 */  { Unit_Volt,         "V",    Unit_Volt,          1, 0 },
/* 20 */  { Unit_MilliVolt,    "mV",   Unit_Volt,      -1000, 0 },
/* 21 */  { Unit_MicroVolt,    "uV",   Unit_Volt,   -1000000, 0 },
/* 22 */  { Unit_Celsius,      "C",    Unit_MilliKelvin, 1000, 273150 },
/* 23 */  { Unit_DeciCelsius,  "dC",   Unit_Celsius, -10, 273150 },
/* 24 */  { Unit_CentiCelsius, "cC",   Unit_Celsius,    -100, 0 },
/* 25 */  { Unit_MilliCelsius, "mC",   Unit_Celsius,   -1000, 0 },
/* 26 */  { Unit_MicroCelsius, "uC",   Unit_Celsius, -1000000, 0 },
/* 27 */  { Unit_Fahrenheit,   "F",    Unit_MilliKelvin, 555, 255116 },
/* 28 */  { Unit_Hertz,        "Hz",   Unit_Hertz,         1, 0 },
/* 29 */  { Unit_KiloHertz,    "kHz",  Unit_Hertz,      1000, 0 },
/* 30 */  { Unit_MegaHertz,    "MHz",  Unit_Hertz,    1000000, 0 },
/* 31 */  { Unit_GigaHertz,    "GHz",  Unit_Hertz, 1000000000, 0 },
};
```

The unit conversion table describes a undirected graph. Each conversion step consists of a scaling factor and a linear offset that will be applied to the original value. By using a depth-first search approach, DCDB can determine a path through this graph that describes how to convert values between various units. The search also takes into account that a conversion step might be applied backwards.

Access to DCDB's unit conversion has been built into `dcdbquery`. To invoke unit conversion in dcdbquery, users can specify a public sensor name followed by a forward slash character and the target unit. For example the command

```
dcdbquery node001_cpu0_temp/K now-1h now
```

will retrieve all values from the last hour of the `node001_cpu0_temp` sensor in Kelvin.

An alternate and simpler approach for value scaling besides the unit conversion framework is the field `scaling_factor` in the column family `publishedsensors`. Whenever this field contains a value, it will be multiplied with the raw sensor value upon readout.

4.7.4 *Integrable Sensors*

The `publishedsensors` column family also contains a Boolean field `integrable`. With this field, users can specify whether the sensor's data can be integrated over time. The common usage for this feature within DCDB is to integrate sampled values of a power sensor measured in Watts into energy measured in Joules. For published sensors with this flag set, DCDB provides the `dcdbquerysum` command line utility. Instead of returning the entire time series for a sensor, `dcdbquerysum` integrates the sensor's values over time in seconds and outputs the result as a single number.

4.7.5 *Virtual Sensors*

DCDB supports the creation of *virtual sensors*. A virtual sensor is a special type of published sensor defined by an arithmetic expression that consists of one or more published sensors.

Exemplary use cases for virtual sensors in the context of energy efficient high performance computing are:

- Calculating the total electric power used by the high performance computer by summing up the values of individual power meters.

- Deriving the average temperature of a compute node from multiple temperature sensors within the node.
- Computing the efficiency of a switched mode power supply as the fraction of AC power input and DC power output.

From a user's perspective, a virtual sensor behaves just like any non-virtual published sensor. Since virtual sensors are published sensors, the creation of a virtual sensor is done similarly to any published sensor by adding an entry to the `publishedsensors` column family. The `publishedsensors` column family provides a Boolean field `virtual`, which will be set to `true` for virtual sensors and which remains unset or is set to `false` for regular published sensors. For all virtual sensors, the `expression` field in the `publishedsensors` column family holds a string describing the arithmetic expression to be evaluated for calculating the virtual sensor's value. As of now, the grammar for arithmetic expression supports the four basic arithmetic operations (`+`, `-`, `*`, `/`), round brackets for operation precedence, and numeric constants. Also, the grammar supports the use of any published sensor's name (virtual or non-virtual) as variable in the arithmetic expression.

For physical sensors, DCDB follows a push approach, meaning that it is at the sensor source's discretion when to send the next data point. Therefore, DCDB has no influence on the time and frequency of incoming sensor data for physical sensors. When reading the data, DCDB simply retrieves all data points in the given time frame. For virtual sensors, however, DCDB needs to be configured in order to know at which time stamps the virtual sensor is supposed to return data during a query. For simplicity, DCDB assumes that a virtual sensor provides data at strict, regular intervals. Hence, the `publishedsensors` column family features two columns: `frequency` and `tzero`. The `frequency` field contains the time between two readings of the virtual sensor in nanoseconds. With the frequency being defined, DCDB also needs information on the phase in order to fully define the time series for a virtual sensor. Thus, the `tzero` field holds the exact time stamp for one of the virtual sensor's readings.

Evaluating the arithmetic expression for a virtual sensor at a given time is trivial, when all inputs into the expression are either numeric constants or are physical sensors for which a reading exists at the exact same time stamp. Unfortunately, however, with DCDB's temporal granularity of one nanosecond, this is rarely the case. Therefore, whenever DCDB evaluates a virtual sensor's value at a given time, a linear interpolation between the neighboring data points on the physical sensors is performed.

5 TEST PLATFORMS

In September 2010, the European Commission opened call FP7-2011-ICT-7 under its 7th Framework Programme for Research, Technological Development and Demonstration Activities. This call included the specific topic ICT-2011.9.13 ("Exa-scale computing, software and simulation") which requested for project proposals developing "advanced computing platforms with potential for extreme performance (100 petaflop/s in 2014 with potential for exascale by 2020)" [69]. After evaluation, the European Commission decided to fund three projects. Two of the three projects, namely DEEP and Mont-Blanc, had a strong hardware focus and delivered fully operational prototype systems. The third project (CRESTA) focused more on application code modernization for Exascale.

Under the assumption that scalability in the system monitoring framework will be essential for future generations of high performance computers, the DEEP and Mont-Blanc projects were considered ideal candidates for installing and evaluating DCDB in practice. This chapter will introduce the two projects, their prototype systems and explain the use of DCDB within the systems.

5.1 MONT-BLANC PROTOTYPE

The Mont-Blanc Project was driven by the idea to leverage technologies originating from the mobile and embedded domains for high performance computing. The rationale behind this approach is that chips for mobile and embedded devices provide high power efficiency at relatively low cost.

To prove the concept, the Mont-Blanc project started its work in 2011 and delivered a fully working prototype in 2015. In addition to the hardware developments, the Mont-Blanc project released a full HPC software stack for the ARM architecture including debuggers and performance analysis tools. Finally, a total of 11 scientific applications covering a wide range of scientific domains were ported and evaluated on the machine.

5.1.1 *System Overview*

The Mont-Blanc prototype consists of 1080 compute nodes in the form of a pluggable edge card [70]. Each of these so called *Samsung Daughter Board* (SDB) cards is powered by a Samsung Exynos 5 Dual 5250 System-on-Chip (SoC). The Exynos 5 SoC integrates two ARM Cortex-

A15 general purpose compute cores at 1.7 GHz and a ARM Mali T-604 Graphics Processing Unit (GPU). Local memory on each SDB is provided by four onboard 1 GB LPDDR4 RAM modules. Since the Exynos 5 SoC does not include on-chip support for external networking, the SDBs also contain a USB 3.0 to Gigabit Ethernet controller. A microSD card on the SDB contains the system's boot image and provides temporary scratch storage.

For physical integration, the Mont-Blanc prototype provides a *blade* architecture. Each blade consists of a so called *Ethernet Mother Board* (EMB) which is a circuit board with connectors for up to 15 SDBs. The EMB has an on-board Gigabit Ethernet / 10-Gigabit Ethernet switch to provide Gigabit network connectivity among the SDBs as well as two 10-Gigabit up-links to the system's central networking switches. Besides, the EMB provides electrical power to the SDBs and implements a detailed power monitoring system capable of capturing high-frequency information of the power delivered to each SDB. To handle the stream of data originating from the high frequency power meters, a FPGA collects and averages the data before forwarding it to the board management controller on the EMB. In addition to the EMB, the blade contains actively managed fans for air cooling and a backplane connector for receiving power from the chassis.

The Mont-Blanc *chassis* is a container for up to 9 Mont-Blanc blades. Each chassis provides a central AC/DC power conversion for its contained blades. It also contains a chassis management controller (CMC) for remote power on/off and an Ethernet switch for the management network that connects the BMCs and the chassis management controller to the central cluster management servers.

Including two additional storage nodes and the central switches for the compute and management networks, the entire Mont-Blanc system fits into two standard 19" 42U cabinets.

5.1.2 *DCDB Monitoring Setup*

This section summarizes the sensors covered by DCDB on the Mont-Blanc prototype system. A more detailed technical description of the setup is given in [3] and [4].

The DCDB installation on the Mont-Blanc prototype system collects information at SDB, EMB, and chassis levels. Table 5 lists all sensors that are collected by DCDB.

Collection of sensors at chassis and EMB level is straightforward since the CMC and the BMC export their sensors as regular IPMI sensors. As such, the IPMI Pusher can be used to collect the data and forward it into DCDB. On the SDB level, the collection of the data generated by the temperature sensor within the Samsung Exynos 5 SoC is also straightforward since the temperature is exported to sysfs. For

Scope	*Sensor Name*	*Readout Path*	*Description*
SDB	PWR	Out-of-Band (via BMC, Mont-Blanc Pusher)	SDB Power Consumption (mW)
SDB	TMP	In-Band (File Pusher)	SDB SoC Temperature (°C)
EMB	PWR	Out-of-Band (via BMC, IPMI Pusher)	EMB Power Consumption (W)
EMB	TMP	Out-of-Band (via BMC, IPMI Pusher)	EMB Temperature (°C)
EMB	PQIITMP	Out-of-Band (via BMC, IPMI Pusher)	PQII Temperature (°C)
EMB	FAN1A_BMC	Out-of-Band (via BMC, IPMI Pusher)	Fan 1A Speed (rpm)
EMB	FAN1B_BMC	Out-of-Band (via BMC, IPMI Pusher)	Fan 1B Speed (rpm)
EMB	FAN2A_BMC	Out-of-Band (via BMC, IPMI Pusher)	Fan 2A Speed (rpm)
EMB	FAN2B_BMC	Out-of-Band (via BMC, IPMI Pusher)	Fan 2B Speed (rpm)
EMB	FAN3A_BMC	Out-of-Band (via BMC, IPMI Pusher)	Fan 3A Speed (rpm)
EMB	FAN3B_BMC	Out-of-Band (via BMC, IPMI Pusher)	Fan 3B Speed (rpm)
EMB	FAN4A_BMC	Out-of-Band (via BMC, IPMI Pusher)	Fan 4A Speed (rpm)
EMB	FAN4B_BMC	Out-of-Band (via BMC, IPMI Pusher)	Fan 4B Speed (rpm)
Chassis	DRAWERPOWER	Out-of-Band (via CMC, IPMI Pusher)	Total Chassis Power Consumption (W)
Chassis	CMBTEMP	Out-of-Band (via CMC, IPMI Pusher)	Chassis Management Board Temperature (°C)
Chassis	FPTEMP	Out-of-Band (via CMC, IPMI Pusher)	Front Panel Temperature (°C)
Chassis	FAN1A_CMC	Out-of-Band (via CMC, IPMI Pusher)	Fan 1A Speed (rpm)
Chassis	FAN1B_CMC	Out-of-Band (via CMC, IPMI Pusher)	Fan 1B Speed (rpm)
Chassis	FAN2A_CMC	Out-of-Band (via CMC, IPMI Pusher)	Fan 2A Speed (rpm)
Chassis	FAN2B_CMC	Out-of-Band (via CMC, IPMI Pusher)	Fan 2B Speed (rpm)
Chassis	PSU1_VIN	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 1 Voltage In (V)
Chassis	PSU2_VIN	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 2 Voltage In (V)
Chassis	PSU3_VIN	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 3 Voltage In (V)
Chassis	PSU4_VIN	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 4 Voltage In (V)
Chassis	PSU1_IIN	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 1 Current In (A)
Chassis	PSU2_IIN	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 2 Current In (A)
Chassis	PSU3_IIN	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 3 Current In (A)
Chassis	PSU4_IIN	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 4 Current In (A)
Chassis	PSU1_PWRIN	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 1 Power In (W)
Chassis	PSU2_PWRIN	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 2 Power In (W)
Chassis	PSU3_PWRIN	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 3 Power In (W)
Chassis	PSU4_PWRIN	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 4 Power In (W)
Chassis	PSU1_12VOUT	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 1 12V Voltage Out (V)
Chassis	PSU2_12VOUT	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 2 12V Voltage Out (V)
Chassis	PSU3_12VOUT	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 3 12V Voltage Out (V)
Chassis	PSU4_12VOUT	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 4 12V Voltage Out (V)
Chassis	PSU1_3v3OUT	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 1 3.3V Voltage Out (V)
Chassis	PSU2_3v3OUT	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 2 3.3V Voltage Out (V)
Chassis	PSU3_3v3OUT	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 3 3.3V Voltage Out (V)
Chassis	PSU4_3v3OUT	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 4 3.3V Voltage Out (V)
Chassis	PSU1_12IOUT	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 1 12V Current Out (A)
Chassis	PSU2_12IOUT	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 2 12V Current Out (A)
Chassis	PSU3_12IOUT	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 3 12V Current Out (A)
Chassis	PSU4_12IOUT	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 4 12V Current Out (A)
Chassis	PSU1_3v3IOUT	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 1 3.3V Current Out (A)
Chassis	PSU2_3v3IOUT	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 2 3.3V Current Out (A)
Chassis	PSU3_3v3IOUT	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 3 3.3V Current Out (A)
Chassis	PSU4_3v3IOUT	Out-of-Band (via CMC, IPMI Pusher)	Power Supply 4 3.3V Current Out (A)

Table 5: Sensors of the Mont-Blanc prototype system collected by DCDB

historic reasons, the respective sysfs entry is read by the File Pusher instead of the sysfs Pusher.

In contrast to the standard implementation of the chassis and EMB sensors via IPMI, the per-SDB power measurement in the Mont-Blanc prototype is a custom development for the project. It features high sampling rates to detect even short spikes in the power trace. In addition, it provides synchronized measurements for better analysis of parallel applications.

To achieve this, the EMB carries one power measurement IC with an I^{2}C interface per SDB slot. Through this I^{2}C interface, it is possible to acquire one power reading every 70 ms. Yet, such a power reading is the result of averaging over 128 individual voltage and current samples performed by the IC.

For generating synchronized readings, the I^{2}C links of the power measurement ICs are all routed into a FPGA which ensures that the I^{2}C commands to trigger the power readings and the processing of the results are performed in parallel. Since the resulting stream of data is too much for the BMC to handle, the FPGA averages the results of 16 consecutive readings per SDB. The FPGA then implements a FIFO buffer which is read periodically by the BMC. The BMC itself uses a 4 MB FIFO buffer in its DRAM to store the data received from the FPGA until the data is read by the user using custom IPMI commands. Since the transfers from the FPGA to the BMC and from the BMC to the user happen in larger packets, the throughput of this setup is significantly higher compared to reading single values using standard IPMI sensor commands. The BMCs have a real-time clock that is synchronized via the Network Time Protocol (NTP). Therefore, the data delivered by the BMC also includes the time of the reading in addition to the power measurement data.

To capture the data blocks containing the power readings from the BMCs via IPMI custom commands, a special pusher had to be developed: the *Mont-Blanc Pusher*. The Mont-Blanc Pusher connects to the BMCs via IPMI and issues all commands required to start the power measurement process and to read the blocks with SDB power data back. Using the time and sensor data from those data blocks, the Mont-Blanc Pusher then forwards all the readings to the Collect Agent in the form of MQTT messages.

Field Width	8 bit	8 bit	16 bit	8 bit	8 bit	8 bit	8 bit
Content	Data Center ID	Cluster ID	Rack ID	Sub-Rack ID	BMC ID	SDB ID	Sensor Type

Figure 8: Definition of the 64-bit Device Location field in the DCDB Sensor ID on the Mont-Blanc prototype

Despite the fact that more than 2800 different sensors are monitored with DCDB on the Mont-Blanc prototype, it is still possible to run DCDB with a single Collect Agent and a single Cassandra instance only. Both components are being run on an external x86-based management server. Yet, the DCDB setup on the Mont-Blanc system already uses an internal sensor naming scheme that allows for scaling up the number of Cassandra instances at any time. For this, the location field of the Sensor IDs used in Mont-Blanc is used as shown in Figure 8. Through this hierarchical splitting among clusters, racks, sub-racks (i.e. chassis), etc. it becomes possible to split the sensor data onto different Cassandra instances based on the sensor's location. For example, one could configure a setup with one Cassandra instance per rack. Also, splitting among groups of racks would be possible since the split onto different Cassandra instances can be done at arbitrary locations in the 128-bit keyspace.

5.2 DEEP PROTOTYPE

The Dynamical Exascale Entry Platform (DEEP) Project was motivated by the assumption that many scientific applications consist of multiple parts with different characteristics. On one side, many applications have code parts with reduced scalability. Those parts benefit from superscalar processors with a high single-thread performance and a flexible network topology. On the other side, application parts with high scalability can benefit from high core counts and often exhibit simpler communication patterns. For this reason, the DEEP project introduced the *Cluster-Booster* concept [71]. The idea behind the Cluster-Booster concept is that a high performance computer consists of two parts. A cluster part for the codes parts with low and medium scalability and a booster part for the highly scalable code parts. As opposed to the traditional approach of a cluster with accelerators, the Cluster-Booster allows for a flexible allocation of cluster and booster resources for each application.

In addition to a standard 128-node computer cluster, the DEEP project delivered three prototype systems:

THE AURORA BOOSTER is the largest of the three systems and also the actual testbed for DCDB. It is powered by 384 Intel® Xeon Phi™ 7120X many core compute accelerators. The cards are connected via an EXTOLL 3D torus network implemented on Altera Stratix V FPGAs. While the Xeon Phi 7120 was originally designed as a PCIe based co-processor accelerator card, the DEEP project developed a solution to have the Xeon Phi act autonomously similar to a regular compute node.

THE GREENICE BOOSTER is a smaller system comprising of 32 Intel® Xeon Phi™ 7120D cards. In this system, the EXTOLL implementation is provided by EXTOLL's "Tourmalet" ASIC for improved networking performance.

THE ENERGY EFFICIENCY EVALUATOR is a miniature replica of the DEEP Cluster and Aurora Booster. It consists of 4 cluster nodes and 16 booster nodes and it's main purpose was to conduct experiments in the field of direct liquid cooling.

Similar to the Mont-Blanc project, the DEEP project brought along six scientific applications to test the new platform in practice. Also, the project delivered a full system software stack including software to boot the Xeon Phi cards over the EXTOLL network and a software router to route MPI data transparently between the InfiniBand based cluster and the EXTOLL based booster parts. In addition, the project conducted research in the field of programming models and performance analysis and prediction tools.

5.2.1 *System Overview*

The DEEP Aurora Booster is fully integrated system hosting all its 384 nodes in a single 21" rack. The Xeon Phi™ 7210X Coprocessor is a 61-core CPU operating at a base frequency of 1.24 GHz. In addition to the processor, the PCIe based accelerator card contains 16GB of main memory and a system management controller (SMC) for out-of-band management.

The usage scenario foreseen by Intel for the Xeon Phi™ x100 family of coprocessors is to mount the PCIe cards into a standard server and to use it in parallel to the host's CPU. In such a setup, the host processor provides a PCIe root complex and the coprocessor acts as a slave on the PCIe bus. Through memory mapped I/O, the host processor can supply the boot image to the coprocessor and initiate the boot process. Even after booting, the accelerator card remains a slave device in the system that is managed by the host.

In the DEEP Aurora Booster, the PCIe lanes of the Xeon Phi™ card are connected to an FPGA implementing the PCIe root complex and an EXTOLL interface. This allows for transferring PCIe packets over the EXTOLL network to the accelerator card. Using this PCIe over EXTOLL setup, the boot image for the coprocessor is transferred and the boot process is initiated. During booting, however, the coprocessor loads a driver for the EXTOLL network. On the PCIe level, the FPGA remains the root complex and the coprocessor remains a PCIe device. Yet, the coprocessor can now send and receive packets on the EXTOLL network as if it was an autonomous compute node.

For physical integration, a single PCB with two FPGAs connects to two Xeon Phi™ cards. This assembly is called a *Booster Node Card* (BNC). In addition to the FPGAs and the coprocessor cards, the BNC carries a board-management controller for remote management as well as temperature and power sensors. The BNC plugs into a backplane that delivers power, EXTOLL networking, and Ethernet networking for the BMCs. Each backplane can host up to eight BNCs. With two backplanes per chassis, a 21" DEEP chassis contains 16 BNCs with a total of 32 booster nodes. The entire Aurora Booster is built as a rack with six chassis on each side for a total of 12 chassis. Since all components of the Aurora Booster are liquid cooling using aluminum cold plates, the chassis also features a cooling liquid distribution bar with quick disconnect couplings for the cooling water. The mechanics of the chassis ensure that a BNC be hot-plugged with the cooling water couplings and the electrical backplane connectors connecting at the same time.

For bridging between the Booster's EXTOLL fabric and the Cluster's InfiniBand fabric, each chassis also contains two *Booster Interface Cards* (BIC). The BICs are powered by an Intel® Xeon™ E3 CPU. Using a PCIe switch, the CPU is connected to an InfiniBand host fabric adapter and an FPGA implementing the EXTOLL fabric. Being connected to both networking fabrics, the BICs' task is to route the networking traffic between the Cluster and the Booster. The second task of the BICs is to take the role of the host processor in booting of the Booster nodes using PCIe over Extoll.

5.2.2 *DCDB Monitoring Setup*

The DEEP Aurora Booster is a useful test environment for DCDB for multiple reasons:

- Central components such as the BNC have been under the control of the project. This includes in particular the firmware for the BMCs on the BNCs. Thus, in DEEP it was possible to implement an actual push model for high frequency power measurement data that sends MQTT messages directly to the Collect Agent. This means that capturing data from certain sensors no longer requires a pusher application as a workaround.

- Being a liquid cooled system, the DEEP Aurora Booster integrates facility side sensors that were installed solely for the project. The integration of those sensors in the DCDB setup is a good proof of concept for a HPC system monitoring solution that integrates both, HPC hardware and facility side sensor information.

- For a true distributed DCDB setup, the 24 BICs in the system are an ideal candidate to run multiple instances of Cassandra on them. For this reason, the BICs were equipped with local solid state drives to store the sensor database.

A thorough description of the DCDB setup on the DEEP Aurora Booster is given in [9] and [10].

Sensor Name	*Readout Path*	*Description*
knc[0,1]-curcore	In-Band (sysfs Pusher)	Processor Core current (mA)
knc[0,1]-curmem	In-Band (sysfs Pusher)	Memory current (mA)
knc[0,1]-curuncore	In-Band (sysfs Pusher)	Un-core current (mA)
knc[0,1]-pwrc2x3	In-Band (sysfs Pusher)	Power delivered via 2x3 12V connector (mW)
knc[0,1]-pwrc2x4	In-Band (sysfs Pusher)	Power delivered via 2x4 12V connector (mW)
knc[0,1]-pwrcore	In-Band (sysfs Pusher)	Core power (mW)
knc[0,1]-pwrimax	In-Band (sysfs Pusher)	Maximum total instantaneous power monitored (mW)
knc[0,1]-pwrinst	In-Band (sysfs Pusher)	Total instantaneous power (mW)
knc[0,1]-pwrmem	In-Band (sysfs Pusher)	Memory power (mW)
knc[0,1]-pwrpcie	In-Band (sysfs Pusher)	Power delivered via PCIe 12 V lines (mW)
knc[0,1]-pwrtot0	In-Band (sysfs Pusher)	Total power (averaged using window 0) (mW)
knc[0,1]-pwrtot1	In-Band (sysfs Pusher)	Total power (averaged using window 1) (mW)
knc[0,1]-pwruncore	In-Band (sysfs Pusher)	Un-core power (mW)
knc[0,1]-tempboard	In-Band (sysfs Pusher)	PCB temperature (°C)
knc[0,1]-tempdie	In-Band (sysfs Pusher)	CPU die temperature (°C)
knc[0,1]-tempgddr	In-Band (sysfs Pusher)	Temperature close to memory (°C)
knc[0,1]-tempinlet	In-Band (sysfs Pusher)	PCB temperature, left (°C)
knc[0,1]-tempoutlet	In-Band (sysfs Pusher)	PCB temperature, right (°C)
knc[0,1]-tempvrcore	In-Band (sysfs Pusher)	VR core temperature (°C)
knc[0,1]-tempvrmem	In-Band (sysfs Pusher)	VR memory temperature (°C)
knc[0,1]-tempvruncore	In-Band (sysfs Pusher)	VR un-core temperature(°C)
knc[0,1]-freq	In-Band (sysfs Pusher)	CPU clock frequency (Hz)
pwr	Out-of-Band (Push from BMC)	Power consumption of EXTOLL FPGAs and BMC
pwr0	Out-of-Band (Push from BMC)	Power consumption of KNC0 (mW)
pwr1	Out-of-Band (Push from BMC)	Power consumption of KNC1 (mW)
tmp0	Out-of-Band (Push from BMC)	Temperature on KNC0 side of FPGA board (°C)
tmp1	Out-of-Band (Push from BMC)	Temperature on KNC1 side of FPGA board (°C)
hum	Out-of-Band (Push from BMC)	Humidity of ambient air at FPGA board (rel. %)

Table 6: Sensors of the DEEP prototype system's BNC related sensors collected by DCDB

As explained in the previous section, at the heart of the DEEP Aurora Booster is the Booster Node Card. The two Xeon Phi™ cards within the BNC already provide a large selection of temperature and power sensors. In addition, the board hosting the two EXTOLL FPGAs and the BMC has been equipped with additional temperature and power sensors as well as a humidity sensor. Table 6 provides an overview of all sensors within the BNC that are covered by DCDB.

The Xeon Phi™ x100 series of co-processor cards was codenamed "Knight's Corner" (KNC). Thus, the sensors have been prefixed with *knc0-* or *knc1-*, depending on which of the two cards inside a BNC they cover. Since the Xeon Phi™ cards are not connected with an out-of-band link for system monitoring, sensors on the cards are captured in-band by the sysfs pusher running on the Xeon Phi™ processor. The pusher uses EXTOLL's IP-over-EXTOLL implementation to establish an MQTT connection to the nearest broker. Unfortunately, this solution implies that the system monitoring infrastructure and the user application running on the DEEP Booster compete for bandwidth on the EXTOLL link. Also, the sysfs pusher uses CPU cycles on the Xeon Phi™ processor that are not available for the user application. In order to mitigate this effect, the sysfs pusher on the DEEP Booster is configured to provide data only at low frequency (<1 Hz).

For the temperature sensors on the Xeon Phi™ cards, low readout frequencies are deemed sufficient as temperature changes do not occur too quickly. Regarding the power sensors, however, higher frequencies might be desirable. Thus, the decision was made to implement additional power sensors with out-of-band access. The additional sensors are mounted on the FPGA-board in the BNC at the 12V supply rail to each Xeon Phi™ card. The sensors consist of Hall effect based current meters and a separate voltage measurement circuitry connected the BMC of the BNC. After calibration of the current sensors, the BMC provides accurate, high-frequency power measurements on a per-Xeon Phi™ basis.

For the DEEP project, a special firmware was developed to run on the BMC of the BNC. This firmware controls the power-up sequence of the BNC including the calibration of the zero-offset for the Hall effect based current sensors at a time when the Xeon Phi™ cards are not powered up. Also, the firmware provides means for updating the bit file of the EXTOLL FPGAs for convenient in-field upgrades of the EXTOLL networking infrastructure. Most importantly, however, the firmware implements a full MQTT pusher to send power data to the nearest Collect Agent. For this, a separate process on the BMC samples the current from the Hall effect sensor as well as the voltage on the 12V rail at high frequency (~100 Hz). The BMC then constructs the appropriate MQTT message for DCDB and sends it to the Collect Agent. The power measurement in the BMC happens completely out-

Sensor Name	*Readout Path*	*Description*
Infra_Aussentemperatur	Out-of-Band (SNMP Pusher)	Outside air temperature (1/10°C)
Infra_Druck_Booster_Sekundaer	Out-of-Band (SNMP Pusher)	Water pressure booster sec. loop
Infra_Druck_Cluster_Sekundaer	Out-of-Band (SNMP Pusher)	Water pressure cluster sec. loop
Infra_Status_Booster_Pumpe	Out-of-Band (SNMP Pusher)	Booster loop pump error state
Infra_Status_Cluster_Pumpe	Out-of-Band (SNMP Pusher)	Cluster loop pump error state
Infra_Status_Dampfbefeuchter	Out-of-Band (SNMP Pusher)	Room air humidifier operational state
Infra_Status_Klimaschrank_Filter	Out-of-Band (SNMP Pusher)	Room air filter error state
Infra_Status_Trockenkuehler	Out-of-Band (SNMP Pusher)	Dry-cooler error state
Infra_Stellsignal_Cluster_Pumpe	Out-of-Band (SNMP Pusher)	Enable signal for Cluster loop pump
Infra_Stellsignal_Klimaschrank	Out-of-Band (SNMP Pusher)	Enable signal for CRAC unit
Infra_Stellsignal_Trockenkuehler_Pumpe	Out-of-Band (SNMP Pusher)	Enable signal for dry cooler loop pump
Infra_Temp_Booster_Austritt	Out-of-Band (SNMP Pusher)	Booster outlet temp pri. loop (1/10°C)
Infra_Temp_Booster_Eintritt	Out-of-Band (SNMP Pusher)	Booster inlet temp pri. loop (1/10°C)
Infra_Temp_Booster_Ruecklauf_Sekundaer	Out-of-Band (SNMP Pusher)	Booster outlet temp sec. loop (1/10°C)
Infra_Temp_Booster_Vorlauf_Sekundaer	Out-of-Band (SNMP Pusher)	Booster inlet temp sec. loop (1/10°C)
Infra_Temp_Cluster_Austritt	Out-of-Band (SNMP Pusher)	Cluster outlet temp pri. loop (1/10°C)
Infra_Temp_Cluster_Eintritt	Out-of-Band (SNMP Pusher)	Cluster inlet temp pri. loop (1/10°C)
Infra_Temp_Cluster_Ruecklauf_1	Out-of-Band (SNMP Pusher)	Cluster outlet temp sec. loop 1 (1/10°C)
Infra_Temp_Cluster_Ruecklauf_2	Out-of-Band (SNMP Pusher)	Cluster outlet temp sec. loop 2 (1/10°C)
Infra_Temp_Cluster_Ruecklauf_3	Out-of-Band (SNMP Pusher)	Cluster outlet temp sec. loop 3 (1/10°C)
Infra_Temp_Cluster_Ruecklauf_4	Out-of-Band (SNMP Pusher)	Cluster outlet temp sec. loop 4 (1/10°C)
Infra_Temp_Cluster_Ruecklauf_Sekundaer	Out-of-Band (SNMP Pusher)	Cluster outlet (mix) temp sec. loop (1/10°C)
Infra_Temp_Cluster_Vorlauf_Sekundaer	Out-of-Band (SNMP Pusher)	Cluster inlet temp sec. loop (1/10°C)
Infra_Temp_Installationsraum	Out-of-Band (SNMP Pusher)	Computer room ambient air temp (1/10°C)
Infra_Temp_Klimaschrank_Ruecklauf	Out-of-Band (SNMP Pusher)	CRAC air outlet temperature (1/10°C)
Infra_Temp_Klimaschrank_Vorlauf	Out-of-Band (SNMP Pusher)	CRAC air inlet temperature (1/10°C)
Infra_Temp_Trockenkuehler_Austritt	Out-of-Band (SNMP Pusher)	Dry-cooler water outlet temp (1/10°C)
Infra_Temp_Trockenkuehler_Eintritt	Out-of-Band (SNMP Pusher)	Dry-cooler water inlet temp (1/10°C)
Infra_Volumenstrom_Cluster_1	Out-of-Band (SNMP Pusher)	Water flow rate cluster sec. loop 1
Infra_Volumenstrom_Cluster_2	Out-of-Band (SNMP Pusher)	Water flow rate cluster sec. loop 2
Infra_Volumenstrom_Cluster_3	Out-of-Band (SNMP Pusher)	Water flow rate cluster sec. loop 3
Infra_Volumenstrom_Cluster_4	Out-of-Band (SNMP Pusher)	Water flow rate cluster sec. loop 4

Table 7: Sensors of the DEEP prototype system's liquid cooling infrastructure collected by DCDB

of-band over the separate management Ethernet network. Thus, it does not interfere with the user application on the Booster at all.

Besides the custom monitoring on the Booster, DCDB also monitors the sensors for the liquid cooling infrastructure on the DEEP prototype system. The liquid cooling infrastructure consists of an outside dry-cooler and an associated water loop. Since the outside loops need to contain anti-freeze additives, a heat-exchanger separates the outside loop from the inner loops for Cluster and Booster. The inner loops consist of separate loops for the Cluster and Booster parts of the system. The Cluster loop is further divided into four legs connected to individual inlets in the Cluster rack. For extended tests, the system is also connected to the computer center's central cold-water distribution system as an additional option for recooling besides the dry-cooler. In addition to the water cooling loops, the room is also equipped with a Computer Room Air Conditioner (CRAC) that cools the computer room's air. A distinct humidifier in the room can increase the room air's humidity if necessary. Control of pumps and monitoring of temperatures, water flow rates, and water pressure is performed by a Sauter Modu525 automation station. The automation station is an embedded computer system with sufficient I/O ports to connect to all sensors and actors. For configuration and monitoring, the automation station provides an Ethernet port with a web-based graphical user interface. Integration into larger monitoring setups and building automation systems can be performed with the included BACnet/IP interface.

BACnet is an ASHRAE/ANSI/ISO standard for building automation and control networks [72]. Initially, BACnet was designed to operate on dedicated RS-232 or RS-485 based networking hardware. Therefore, the protocol does not implement additional security measures and anyone with access to the BACnet physical network infrastructure can read and write data from and to BACnet devices. The later adoption of the BACnet/IP standard for transmitting BACnet messages over Ethernet or other IP based networks also did not add any security layer. It is therefore common practice to operate separate networks for building automation to prevent unauthorized access to BACnet devices over the network.

For these security reasons, the automation station in the DEEP system is not connected to DCDB via its Ethernet port directly. Instead, a BACnet/IP to SNMP gateway device has been configured that provides two network interfaces and exposes the sensor data captured on the BACnet interface to SNMP clients on the second network interface. With this setup, the cooling infrastructure related monitoring data becomes available for DCDB using the SNMP Pusher. Table 7 has the list of all infrastructure related sensors that are available to DCDB.

Field Width	8 bit	8 bit	16 bit	8 bit	8 bit	8 bit	8 bit
Content	Data Center ID	Cluster ID	Rack ID	Chassis ID	BIC ID	BNC ID	KNC ID

Figure 9: Definition of the 64-bit Device Location field in the DCDB Sensor ID on the DEEP prototype

Besides the custom BMC firmware supporting the direct push of sensor data and the integration of cooling infrastructure sensors, the DEEP prototype system also serves as an excellent testbed for the distributed nature of DCDB. Due to the high data rate of the power sensors on the BNCs, a single central monitoring server is unlikely to be sufficient. The DCDB setup on the DEEP Aurora Booster therefore uses the BICs for storing the sensor data. As in the Mont-Blanc prototype, the organization of sensors follows a hierarchical approach within the device location field in the internal sensor IDs. In DEEP, the sensor IDs are defined as shown in Figure 9. For configuring a

Hostname	*Chassis ID*	*BIC ID*	*Sensor ID range*
bic001	0	0	`00000000 00000000 00000000 00000000` `- 00000000 0000FFFF FFFFFFFF FFFFFFFF`
bic002	0	1	`00000000 00010000 00000000 00000000` `- 00000000 00FFFFFF FFFFFFFF FFFFFFFF`
bic003	1	0	`00000000 01000000 00000000 00000000` `- 00000000 0100FFFF FFFFFFFF FFFFFFFF`
bic004	1	1	`00000000 01010000 00000000 00000000` `- 00000000 01FFFFFF FFFFFFFF FFFFFFFF`
bic005	2	0	`00000000 02000000 00000000 00000000` `- 00000000 0200FFFF FFFFFFFF FFFFFFFF`
⋮	⋮	⋮	⋮
bic023	11	0	`00000000 0B000000 00000000 00000000` `- 00000000 0B00FFFF FFFFFFFF FFFFFFFF`
bic024	11	1	`00000000 0B010000 00000000 00000000` `- 00000000 0BFFFFFF FFFFFFFF FFFFFFFF`
deepm	n/a	n/a	`00000000 0C000000 00000000 00000000` `- FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF`

Table 8: Configuration of the Cassandra database ring structure on the DEEP Booster

distributed Cassandra setup for DCDB, the partition key ranges for each Cassandra instance need to be defined. This leads to the specification of partition key ranges resulting from the device location scheme as shown in Table 8. Upon closer inspection it becomes clear that the ranges are not equal in size. For example, the range covered by `bic002` is larger than the range covered by `bic001`. This is due to the fact that the configuration ring must specify a storage location for each possible partition key. In the case of the DEEP definition of the sensor location field, this means that BIC ID for a sensor can go as high as 255 and all of the sensors would be mapped the Cassandra instance of the BIC with ID 1. In principal, this is not a problem because the partition key ranges can be split at any time with the insertion of additional monitoring nodes. Even without additional nodes, the system would continue to work as long as the influx of sensor data per Cassandra node can still be handled by the underlying hardware. At the rear end of the partition key mapping, the `deepm` management and login server takes all sensors that are not mapped to any of the BICs, including those of the cooling infrastructure.

6 TESTS & PERFORMANCE OPTIMIZATIONS

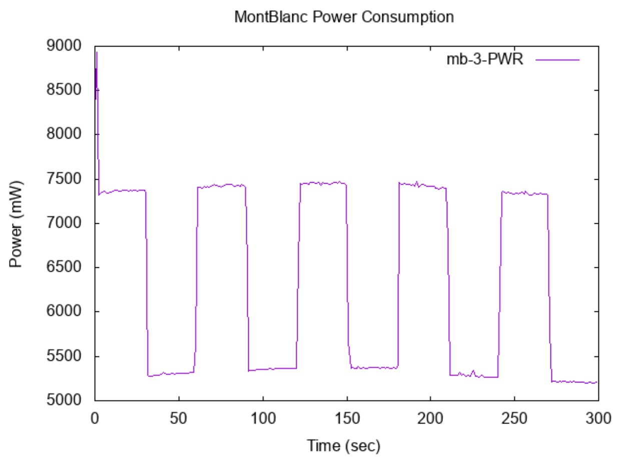
As it has been explained in the previous chapter, the Mont-Blanc and DEEP projects are the first test environments for the DCDB framework. Yet, even before the DCDB installation on the two systems had started, simple tests using artificial monitoring data (e.g. script generated series of values containing data from sine or rectangle functions) have been performed to validate its basic functionality. Once most internals of DCDB were considered stable enough, the software was deployed on the prototype systems. On both systems, the next step consisted of functional tests of the system specific monitoring components. In particular, efforts have been spent on verifying the sensors for power measurement on the two systems.

This chapter highlights some of the tests that were carried out on the two machines in which DCDB played a prominent role. It also summarizes the tests that were performed to demonstrate DCDB's performance. Whenever it is appropriate, an outlook on possible future work will be given in this chapter.

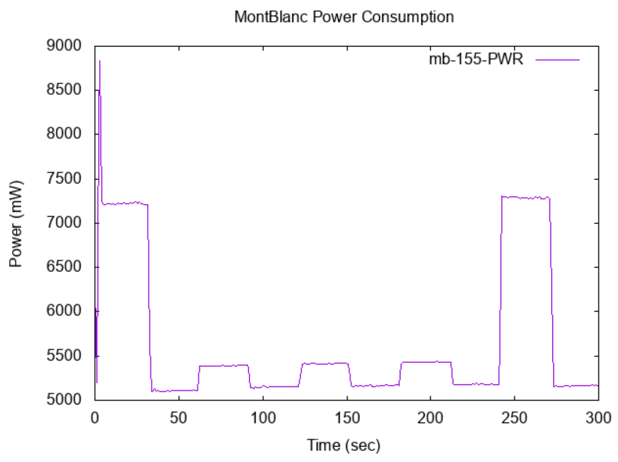
6.1 VERIFYING THE MONT-BLANC POWER MEASUREMENT SETUP

As explained in the previous chapter, the Mont-Blanc system performs power measurements using digital power metering ICs on the EMB. Sensors such as these power metering ICs, just like any other electronic component, are subject to production errors and failure. Thus, the sensors had to be tested before the power monitoring data collected by DCDB could be made publicly available. Two special aspects of the Mont-Blanc monitoring setup were also taken into account. First, the use of a custom data format to transfer multiple power measurement data points at once increases the likelihood of implementation errors (e.g. power data being attributed to the wrong SDB). Second, the BMC on the EMB provides time stamps of the measurement points making it necessary to synchronize the BMC's clock accordingly.

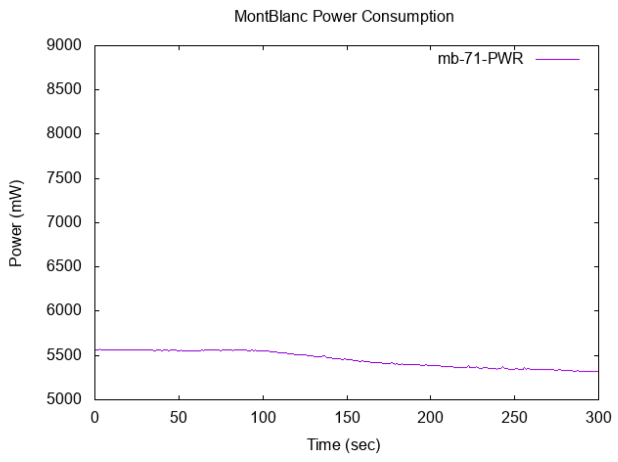
To verify the entire measurement workflow, a small test application was written that was executed on all SDBs, one after another. The test program's purpose was to perform CPU-intensive calculations for 30 seconds, followed by a 30 second period of no activity. This duty-idle power cycle was repeated five times. With the power consumption being high during the duty periods and low in the idle periods, the time series graph of the power meter will resemble a square wave pattern. The rationale behind this approach is that the pattern will be



(a) Expected power consumption pattern during test execution



(b) Reduced power consumption due to thermal throttling



(c) Defective measurement sensor

Figure 10: Power consumption during execution of the power cycle tests on the Mont-Blanc system

prominently visible, even when looking at a longer time series graph. Repeating the cycle five times is done to avoid confusion with other duty-idle periods that could also originate from the execution of an ordinary application.

After the program has run on all SDBs, the start and end times of each job were obtained from the Slurm resource management system. Feeding these into `dcdbquery` returned the power trace for the application run. The graphs of the power traces for each of the 1080 SDBs was then manually inspected. This inspection revealed some BMCs for which NTP time synchronization had not been configured. After correcting the NTP configuration on the BMCs and documenting the faulty measurement IC, all graphs showed the square wave pattern.

Most of the graphs showed a regular pattern for the power consumption of the nodes as shown in Figure 10a. On some nodes, however, the power draw during some or all of the duty cycles was significantly lower than on other nodes. An example of this behavior is shown in Figure 10b. Further investigation revealed that the affected SDBs were running hotter than others resulting in a throttling of the CPU. Following a change in the fan control policy of the chassis, all SDBs were sufficiently cooled and a repetition of the test on all nodes showed the expected behavior with no more anomalies except for one: one defective power measurement IC delivered a near constant reading despite the benchmark being executed without anomalies on the node (Figure 10c). This node was subsequently removed from the queuing system during power consumption related benchmarks.

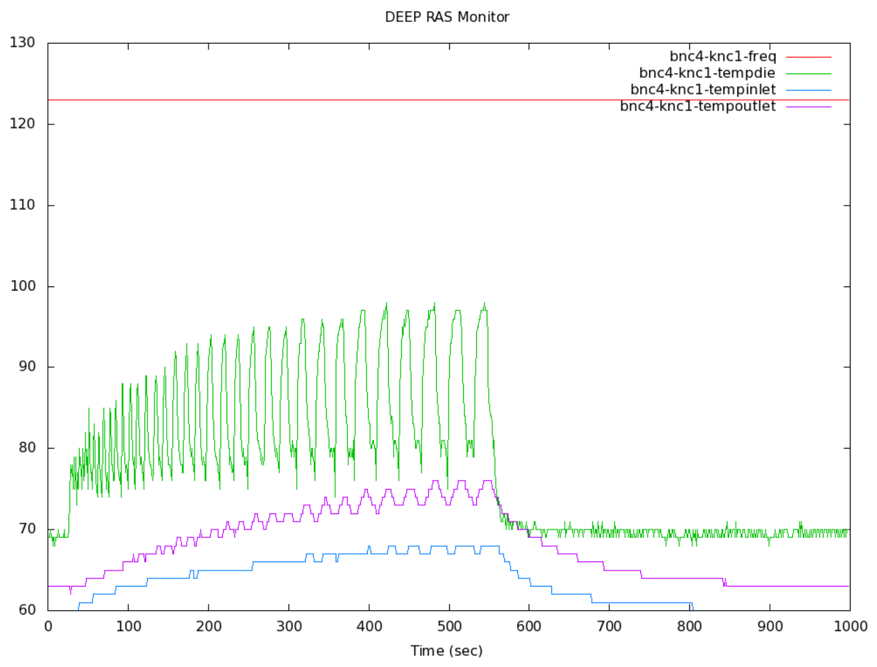
The issues found during the power cycle tests emphasize the importance of thoroughly testing any measuring setup before use instead of trusting the measurement equipment blindly. Also, tracking down the oddities in the power consumption to a thermal root cause is a good argument for integrated setups which combine monitoring information from IT and infrastructure into one monitoring system.

Outlook and ideas for future work

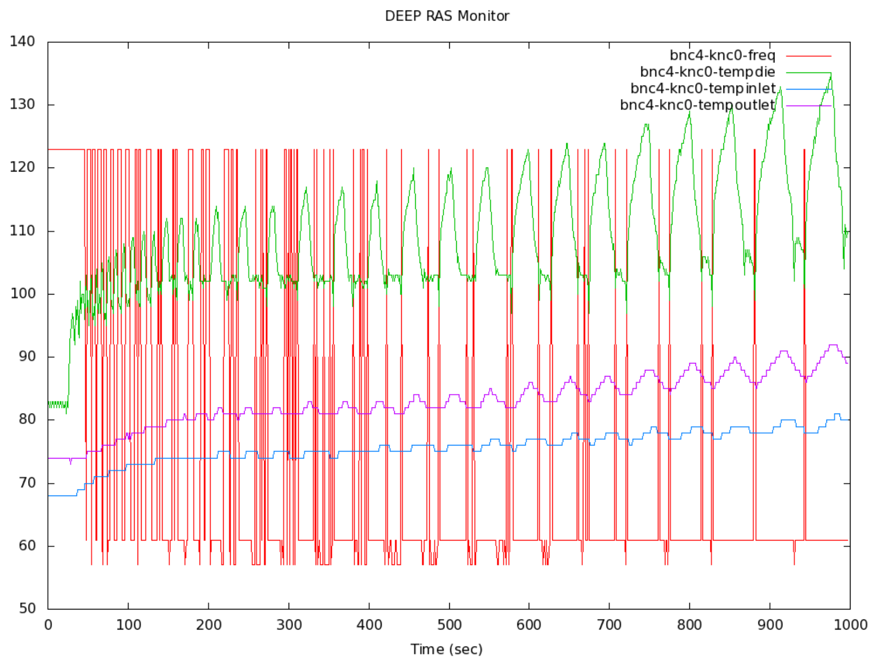
In this series of tests, evaluation of the power consumption graphs was performed manually through human visual inspection. Since this will no longer be feasible for systems that are significantly larger in size, future research could explore possibilities in automating this task. For simple test cases such as the above power cycle application, algorithms from the signal processing domain could yield acceptable results. In case of more complex tests with multiple sensors, machine learning technologies could be used instead.

6.2 THERMAL THROTTLING ON THE DEEP SYSTEM

One part of the evaluation tests on the DEEP system was dedicated to the machine's liquid cooling infrastructure. According to the American Society of Heating, Refrigerating, and Air-Conditioning Engineers (ASHRAE) [73], liquid cooled computers can be categorized



(a) Properly cooled node — no thermal throttling



(b) Node with reduced cooling — CPU enters thermal throttling

Figure 11: Time series of selected temperature sensors and CPU frequency during a HPL run on the DEEP prototype

with respect to their operating temperatures into five different groups (W1 to W5). A system certified for ASHRAE W4 operations can be run with water inlet temperatures of up to 45 °C and is therefore qualified for year-round chiller-less cooling almost anywhere on earth. Systems certified for ASHRAE W5 operation operate at temperatures greater than 45 °C and re-use the excess heat in the central building heating system or for other purposes. The goal for the DEEP system was to at least comply with the W4 standard. Since no upper temperature is defined for systems complying with W5, the tests were also trying to determine the consequences of operating beyond 45 °C.

For this test, a subset of 4 BNCs (8 nodes) was running single-node versions of the high performance Linpack (HPL) benchmark to generate CPU load. The water inlet temperature was then set to values from 20 °C to 50 °C in steps of 5 K. Starting from 40 °C, one of the eight nodes started to show longer runtimes for each iteration of the benchmark and at 45 °C and 50 °C, two nodes of the eight nodes exhibited longer benchmark runtimes than the other nodes.

With DCDB at hand, tracking down the cause for this behavior was easy. In the background, DCDB had collected all information including temperatures and CPU frequency. Figure 11 shows the time series of the CPU frequency (scaled by 1/100), die temperature, and two additional temperature sensors on the board during the HPL run. The trace shows the behavior of the HPL benchmark which was configured to perform multiple iterations on data sets of increasing size. While the CPU frequency remains constant on the unaffected node independent from the problem size (Figure 11a), the slow node could avoid thermal throttling only in the first iterations (Figure 11b). The overall temperature level on the affected node is approximately 5-10 °C higher in idle. Under load, the die temperature of the throttled node exceeds 130 °C whereas the unaffected node's die temperature never exceeds 100 °C. This clearly indicated that the cooling of the affected node needed reworking (i.e. re-mounting of the cold plate). Yet, in general, the BNCs were well suited for operating at cooling water temperatures of even 50 °C.

The conclusion from these tests for the DEEP project was that the system works well within the specifications of ASHRAE W4 and possibly W5. It is also good to see that the CPU does a good job at protecting itself from overheating. The conclusion from a system monitoring perspective was that being able to monitor as many parameters as possible makes analysis and evaluation of high performance computing systems very effective. Without DCDB, additional time-consuming iterations of the tests would have been necessary to analyze the behavior. Thanks to the scalability of DCDB, however, monitoring all parameters at a high temporal resolution was possible and the entire analysis of the situation was possible retrospectively.

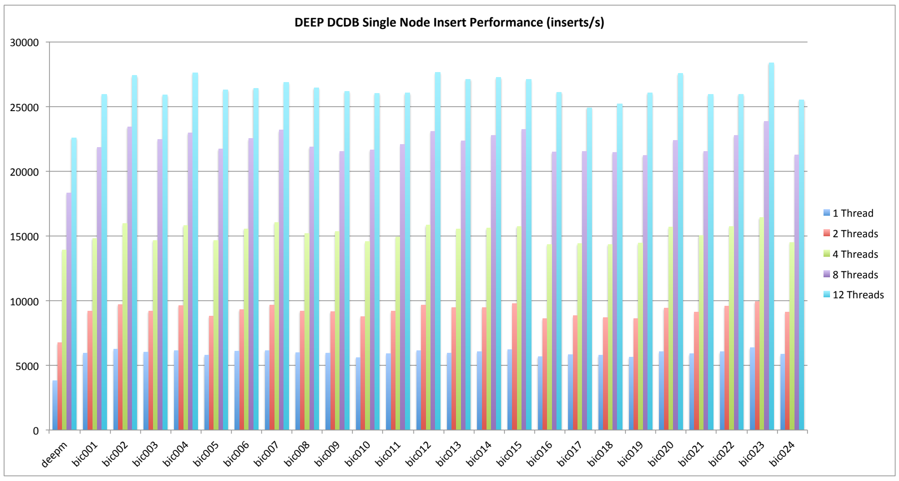
6.3 BENCHMARKING DCDB'S INSERT PERFORMANCE

The design goal for DCDB was to achieve a high insert performance in order to being able to capture and store as much sensor data as possible. Cassandra as the underlying database promises scalability and the data model for storing the sensor data in DCDB can fulfil this promise in theory. Since the entire process of inserting data into DCDB not only concerns Cassandra but also the Collect Agent, it was sensible to test the behavior of DCDB's performance scalability in practice.

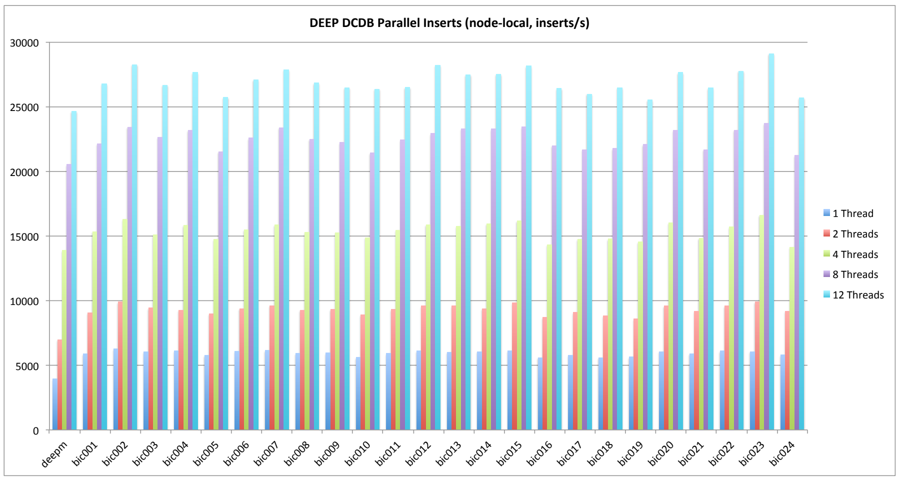
The DEEP Aurora Booster with its Cassandra setup spanning the 24 BICs and the `deepm` management server is a good hardware platform for this test. On the software side, the test setup consists of the standard DCDB installation on the DEEP Aurora Booster and a special benchmark application. This benchmark application mimics a pusher application that sends MQTT messages with sensor data as fast as possible. The benchmark was configured to send data for 24 simulated sensors. For each of the 24 simulated sensors, the benchmark application sends 100,000 MQTT messages. The benchmark waits until each MQTT message is delivered before sending the next message. This means that any bottleneck at the Collect Agent or Cassandra will propagate back to the benchmark application and no data will be lost. Since the Collect Agent and Cassandra operate with multiple threads, the benchmark application is also written to send messages in parallel with a configurable number of threads. For this evaluation, repeated tests were performed with 1, 2, 4, 8, and 12 threads in the benchmark application. The performance evaluation will look at the total number of inserts per second into DCDB per node. To avoid noise in the test setup, the Booster Nodes were shut down during the benchmarks and the benchmark application was started on the Cassandra nodes themselves.

The first test's goal was to create a baseline for the measurements. Therefore, the benchmark was started on each node exclusively, i.e. all nodes except for the node running the benchmark were not handling any sensor data. The results of this test are shown in Figure 12a. The chart shows that all nodes can achieve an insert performance of over 20,000 inserts per second. It also shows that the performance increases with the number of MQTT message threads. While this is only a confirmation of the expected behavior, it confirms that pusher applications handling data from multiple data sources should feature a threaded design.

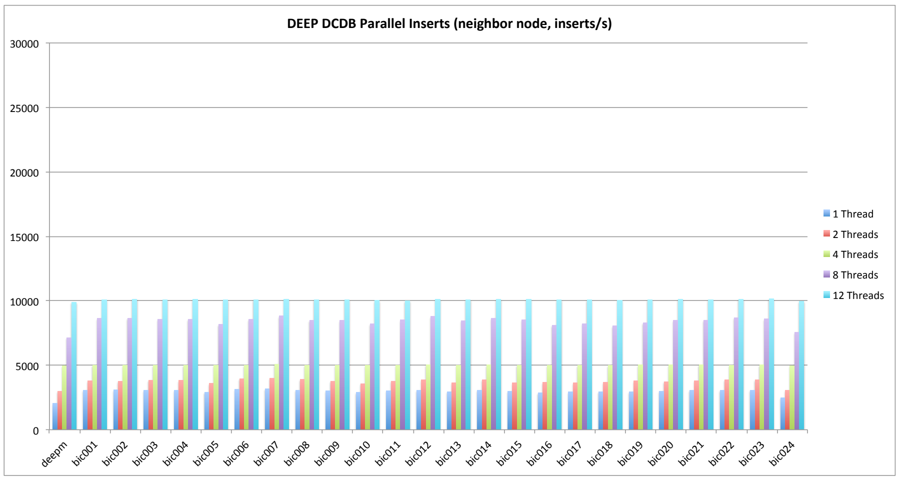
To prove the scalability of the entire solution, the benchmark application was run again. This time, however, it was executed on all nodes in parallel. Given the promise of linear scalability of the entire solution, the expected performance should be similar to the performance observed when running the benchmark node by node. The results in



(a) Performance of single node inserts



(b) Performance of all node parallel inserts



(c) Performance of all node parallel inserts on neighboring nodes

Figure 12: Analysis of the DCDB insert performance on the DEEP Booster.

Figure 12b show that the performance is indeed similar. This result proves that a DCDB installation scales linearly with the number of DCDB nodes running a Cassandra instance and the Collect Agent. As a consequence, apart from the limits imposed by the 64 bits of a sensor's location identifier, DCDB will be capable of monitoring systems of arbitrary size. Should the performance of DCDB saturate (e.g. by pushing too many sensors or by pushing at too high frequencies), the situation can always be relaxed by adding more DCDB nodes.

For maximum performance, it is required to insert data into Cassandra on the Cassandra node that will store the data according to Cassandra's configuration. As explained previously, careful planning during the installation phase of DCDB is required to ensure that this requirement is fulfilled. Cassandra is kind enough, however, to copy data onto the correct node if an insert is performed on a wrong node. Obviously, such copying of data in the background impacts the overall performance of Cassandra by creating additional networking traffic. To estimate the maximum impact of this effect, a third experiment was conducted. Again, the insertion benchmark was run on all nodes in parallel using the smallest possible MQTT messages with a single sensor value. This time, however, the sensor IDs used by the benchmark were not sensor IDs that resulted in Cassandra storing the data on the current node. Instead, each benchmark application created data for sensor IDs that was supposed to be stored on the next node in the Cassandra configuration. For example, the benchmark running on `bic001` created data that was stored on `bic002` and so forth. The results are shown in Figure 12c. As expected, the overhead created through the additional Cassandra-internal transfer of data is significant as the insert performance shrinks to less than half the performance of the optimal setup.

Outlook and ideas for future work

Since wrongly configuring a DCDB installation can have such significant impact on the overall performance, future research could work out ideas for facilitating the installation of DCDB for new users. Possible approaches could involve the design of a proper user interface for installing and configuring DCDB as well as network based auto-discovery mechanisms for sensors and pusher applications to assign suitable sensor IDs and to find the nearest Cassandra node.

6.4 OPTIMIZED VIRTUAL SENSOR EVALUATION

DCDB's virtual sensor feature provides a powerful mechanism to consolidate the data of many physical sensors into aggregated values. With the right physical sensors available inside DCDB, a virtual sensor could even calculate a single data center wide metric, for example PUE. Yet, the evaluation of virtual sensors can encounter scalability limits.

Section 4.7.5 explained that DCDB supports a simple grammar describing the arithmetic for evaluating the virtual sensor at a given timestamp. Whenever the evaluation of a virtual sensor is requested, DCDB creates an abstract syntax tree for the expression. DCDB then recursively processes the syntax tree for each timestamp at which the virtual sensor's value is needed. Once the processing of the syntax tree requires the input of a physical sensor s at time t, DCDB looks up the time series of s to determine the reading (t_1, v_1) prior to t and the reading (t_2, v_2) following t. The value $v(t)$ of the physical sensor at time t is then approximated using linear interpolation:

$$v(t) = \frac{v_2 - v_1}{t_2 - t_1} * t + \frac{t_2 v_1 - t_1 v_2}{t_2 - t_1} \tag{7}$$

Determining the readings (t_1, v_1) and (t_2, v_2) can be done efficiently with Cassandra and DCDB's database scheme. Yet, the above scheme requires $2n$ readings of physical sensor values when evaluating a virtual sensor which depends on n physical sensors as inputs. For virtual sensors whose reporting frequency is similar or greater than the reporting frequency of its underlying physical sensors, the above approach can also lead to repetitive readings of the same value from the data store.

To improve the performance of virtual sensor evaluation, DCDB implements a scheme of sensor data caches. After DCDB has built the abstract syntax tree for a virtual sensor expression, it determines a list of all physical sensors which are needed for evaluation of the virtual sensor. The determination of the physical sensors is performed recursively, taking into account also the cases in which a virtual sensor expression refers to other virtual sensors, which in turn may refer to more virtual or physical sensors. Once the list of physical sensors has been determined, DCDB allocates a sensor data cache for each physical sensor. Requests for physical sensor values as well as the seeking for data points preceding and following a certain timestamp can now be done using the sensor data cache. Whenever a request to the sensor data cache is made, the implementation checks whether the requested value has previously been loaded from the database. If not, the sensor data cache attempts to fetch a certain number of readings prior and after the requested reading form the database and stores the values in memory. The number of readings that are speculatively loaded by the sensor data cache is configurable to allow for application specific tradeoffs between lowering the number of requests to Cassandra and the memory consumption of the sensor data caches.

Outlook and ideas for future work

The virtual sensors feature offers a variety of options for optimization and future research. One idea that has not yet been implemented is a value cache for virtual sensor values. DCDB is built with the assumption that sufficient space for storing all data is available. Thus, one could conclude that it is better to store all calculated virtual sensor values back to the sensor data store so that they do not need to be recalculated when they get queried again.

Obviously, care needs to be taken to delete the values from the data store when the definition of a virtual sensor is changed. Additionally, DCDB would need to address situations, in which a physical sensor reading is inserted after a virtual sensor has been evaluated that would have been based on the physical sensor's reading.

Further research could also address the parallel and distributed evaluation of virtual sensors. For this, the underlying abstract syntax tree of a virtual sensor expression could be reordered with the goal to distribute the evaluation onto the database nodes while increasing data locality at the same time. For example, the calculation of a global sum of power sensors could be performed locally on each Cassandra node for all its local sensors before the partial sums are sent to the requesting node where the final sum is calculated. Reshaping the syntax tree using the commutative and distributive properties of the supported basic arithmetic functions could be used to further optimize such distributed approach.

Finally, the grammar of the expressions for virtual sensors could be improved to implement new features. Possible features could include derivative calculation, helper functions (e.g. to return the number of available sensors matching a given name pattern), or even if-then-else constructs that control the evaluation of each single data point.

7 CONCLUSION & OUTLOOK

In a study performed by Hyperion Research, the market for high performance computing systems is projected to grow from 11.2 billion US Dollars in 2016 to over 14.8 billion Dollars in 2021 [74]. Considering that the ratio of cost per computational capacity has been decreasing throughout the history of computing, one can conclude that the use of supercomputers will continue to grow as a valuable tool for scientists and engineers across a multitude of domains. At the same time, lowering the total cost of operation in high performance computing will remain a means of supercomputing providers and users to further increase their capabilities and to outperform their competition. Similar to commercial data center applications, electricity costs make up for the largest fraction of operational costs in supercomputers and the electricity costs do not only incur in the computer equipment itself, but also in the surrounding data center infrastructure.

While all of this has been well understood and models, criteria, and metrics for the evaluation of energy efficiency in high performance computing have been defined, practical application of these models and the actual determination of the associated metrics remained a challenge. Several reasons for this can be named:

INSUFFICIENT OR IMPROPER INSTRUMENTATION causes a lack of data required for full evaluation or the desired metrics. However, decreasing costs for sensors and measuring equipment alongside a continuous renewal process in the data centers will help overcome these issues with time.

LACK OF INTEGRATION of monitoring systems, in particular the separation of IT monitoring and building infrastructure monitoring systems, make calculating side-wide energy efficiency metrics a tedious task. None of the existing IT monitoring solutions presented in this thesis is ready to connect to building infrastructure management systems.

LIMITED SCALABILITY of monitoring systems is another hurdle for any integrated monitoring approach. If the IT monitoring solution of a large system can barely handle the IT related data, operators will not dare to add the building infrastructure related data to it and vice versa.

This thesis set out to address, at the very low technical level, the shortcomings of today's system monitoring solutions to overcome these limitations. With the Data Center DataBase (DCDB) framework, a scalable approach has been presented for collecting, storing, processing, and reporting time series of sensor data. The scalability of DCDB allows for improving the spatial and temporal resolution of today's data center monitoring solutions to an arbitrary extent. Through the use of simple, lightweight, and standardized interfaces, it is easy to make DCDB the host for all monitoring data in high performance computing data centers, no matter whether it is server monitoring data, building infrastructure data, or application performance monitoring data. With the flexibility and configurability introduced by the implementation of a virtual sensor concept, DCDB can be configured to directly generate any energy efficiency related metric. For example, this means that DCDB can be used to calculate the Power Usage Effectiveness over an entire year of operation, while at the same time providing near real-time data on particular subsystems which can be used by the system management software to (auto-)tune certain hardware parameters for optimized energy efficiency.

DCDB has proved its capability to deliver on these promises in the European DEEP and Mont-Blanc projects. Yet, the use in both projects has also shown that more work is needed to turn DCDB from its proof-of-concept state into a truly valuable tool for high performance computer operators and users. From its current state, DCDB could either be used to replace the data collection and processing infrastructures in existing monitoring tools, or it could be further developed into its own fully integrated monitoring solution.

To encourage the use of DCDB and to allow for a broader community to determine the future of the project, DCDB has been put under the GNU General Public License (GPL v2 for the command line programs and LGPL v2.1 for the DCDB library).

BIBLIOGRAPHY

1. Auweter, A. *et al. Mont-Blanc Project Deliverable 5.3: Preliminary Report on Porting and Tuning of System Software to ARM Architecture* http://montblanc-project.eu/sites/default/files/d5.3_preliminary_report_on_porting_system_software_to_arm_architecture_v1.pdf.
2. Dózsa, G. *et al. Mont-Blanc Project Deliverable 5.5: Intermediate Report on Porting and Tuning of System Software to ARM Architecture* http://montblanc-project.eu/sites/default/files/D5.5_Intermediate_report_on_porting...._v1.0.pdf.
3. Tafani, D. & Auweter, A. *Mont-Blanc Project Deliverable 5.8: Prototype demonstration of energy monitoring tools on a system with multiple ARM boards* http://montblanc-project.eu/sites/default/files/MB_D5.8_Prototype%20demonstration%20of%20energy%20monitoring...v1.0.pdf.
4. Mantovani, F. *et al. Mont-Blanc Project Deliverable 5.11: Final report on porting and tuning of system software to ARM architecture* http://montblanc-project.eu/sites/default/files/D5.11%20Final%20report%20on%20porting%20and%20tuning.%20V1.0.pdf.
5. Tafani, D. & Auweter, A. *Mont-Blanc Project Deliverable 7.4: Concept for energy aware system monitoring and operation* http://montblanc-project.eu/sites/default/files/D7.4_Concept_for_energy_aware_system_monitoring_andoperation_v1_reduced.pdf.
6. Meyer, N., Solbrig, S., Wettig, T., Auweter, A. & Huber, H. *DEEP Project Deliverable 7.1: Data centre infrastructure requirements* http://www.deep-project.eu/SharedDocs/Downloads/DEEP-PROJECT/EN/Deliverables/deliverable-D7.1.pdf.
7. Meyer, N. *et al. DEEP Project Deliverable 7.2: Concepts for improving energy and cooling efficiency* http://www.deep-project.eu/SharedDocs/Downloads/DEEP-PROJECT/EN/Deliverables/deliverable-D7.2.pdf.
8. Auweter, A. & Ott, M. *DEEP Project Deliverable 7.3: Preparation of energy and cooling experimentation infrastructure* http://www.deep-project.eu/SharedDocs/Downloads/DEEP-PROJECT/EN/Deliverables/deliverable-D7.3.pdf.
9. Auweter, A. & Ott, M. *DEEP Project Deliverable 7.4: Intermediate Report on DEEP Energy Efficiency* http://www.deep-project.eu/SharedDocs/Downloads/DEEP-PROJECT/EN/Deliverables/deliverable-D7.4.pdf.
10. Ott, M. & Auweter, A. *DEEP Project Deliverable 7.5: Final Report on DEEP Energy Efficiency* http://www.deep-project.eu/SharedDocs/Downloads/DEEP-PROJECT/EN/Deliverables/deliverable-D7.5.pdf.
11. Auweter, A., Bode, A., Brehm, M., Huber, H. & Kranzlmüller, D. in *Information and Communication on Technology for the Fight against Global Warming* (eds Kranzlmüller, D. & Tjoa, A. M.) 18–25 (Springer, 2011).
12. Wilde, T., Auweter, A. & Shoukourian, H. The 4 Pillar Framework for energy efficient HPC data centers. *Computer Science-Research and Development* **29,** 241–251 (2014).
13. Auweter, A. *et al. A case study of energy aware scheduling on SuperMUC* in *Supercomputing* (eds Kunkel, J. M., Ludwig, T. & Meuer, H. W.) (2014), 394–409.
14. *LRZ Usage Statistics Website* https://www.lrz.de/services/compute/supermuc/statistics/.
15. Joseph, E. C., Conway, S. & Sorensen, R. *Worldwide HPC Server 2015–2019 Forecast* Market Analysis (IDC, 2015).

16. Intersect360. *Worldwide High Performance Computing 2014 Total Market Model and 2015–2019 Forecast* tech. rep. (Intersect360, 2015).

17. *Top500 The List. Website* http://top500.org/.

18. Moore, G. Cramming More Components Onto Integrated Circuits. *Electronics* **38** (Apr. 1965).

19. Beloglazov, A., Buyya, R., Lee, Y. C. & Zomaya, A. in *Advances in computers* 47–111 (Elsevier, 2011).

20. Valentini, G. L. *et al.* An overview of energy efficiency techniques in cluster computing systems. *Cluster Computing* **16,** 3–15 (2013).

21. Pelley, S., Meisner, D., Wenisch, T. F. & VanGilder, J. W. *Understanding and abstracting total data center power* in *Workshop on Energy-Efficient Design* **11** (2009).

22. VDI Verein Deutscher Ingenieure e.V. *VDI-Wärmeatlas* 11. Aufl. 2013. ISBN: 978-3-642-19982-0 (Springer Berlin, Heidelberg, Wiesbaden, 2013).

23. *LRZ SuperMUC Website* https://www.lrz.de/services/compute/supermuc/.

24. Nikolopoulos, D. S. *et al.* Energy efficiency through significance-based computing. *Computer* (ed Helal, S.) 82–85 (2014).

25. Wilde, T. *et al. CooLMUC-2: A supercomputing cluster with heat recovery for adsorption cooling* in *2017 33rd Thermal Measurement, Modeling Management Symposium (SEMI-THERM)* (2017), 115–121. doi:10.1109/SEMI-THERM.2017.7896917.

26. Roy, K., Mukhopadhyay, S. & Mahmoodi-Meimand, H. Leakage current mechanisms and leakage reduction techniques in deep submicrometer CMOS circuits. *Proceedings of the IEEE* **91,** 305–327 (2003).

27. *Green500 List. Website* http://green500.org/.

28. Ge, R., Feng, X., Pyla, H., Cameron, K. & Feng, W. Power measurement tutorial for the Green500 list. *The Green500 List: Environmentally Responsible Supercomputing* (2007).

29. *Energy Efficient High Performance Computing Working Group Website* https://eehpcwg.llnl.gov.

30. The Energy Efficient HPC Working Group. *Energy Efficient High Performance Computing Power Measurement Methodology* Version 2.0 RC1 (2014).

31. Scogland, T. R. *et al. A Power-measurement Methodology for Large-scale, High-performance Computing* in *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering* (eds Lange, K.-D., Murphy, J., Binder, W. & José, M.) (ACM, Dublin, Ireland, 2014), 149–159. ISBN: 978-1-4503-2733-6. doi:10.1145/2568088.2576795. http://doi.acm.org/10.1145/2568088.2576795.

32. Avelar, V., Azevedo, D. & French, A. *PUE™: A Comprehensive Examination of the Metric* tech. rep. (2012).

33. *The Green Grid Website* http://www.thegreengrid.org/.

34. Patterson, M. K. *et al. TUE, a new energy-efficiency metric applied at ORNL's jaguar* in *Supercomputing* (eds Kunkel, J. M., Ludwig, T. & Meuer, H. W.) (2013), 372–382.

35. Wilde, T. *et al. DWPE, a new data center energy-efficiency metric bridging the gap between infrastructure and workload* in *High Performance Computing Simulation (HPCS), 2014 International Conference on* (eds Bassini, S., Nygård, M., Smari, W. & Spalazzi, L.) (July 2014), 893–901. doi:10.1109/HPCSim.2014.6903784.

36. *Nagios Website* https://nagios.org/.

37. *Nagios Enterprises LLC Website* https://nagios.com/.

38. Arraj, V. ITIL™: the basics. *Buckinghampshire, UK* (2010).

39. *Icinga Website* https://icinga.org/ (2014).

40. *Check_MK Website* https://mathias-kettner.de/check_mk.html.

41. Oetiker, T. *RRDtool* http://rrdtool.org/ (2014).

42. *Score-P Website* http://score-p.org/.

43. Geimer, M. *et al.* The Scalasca performance toolset architecture. *Concurrency and Computation: Practice and Experience* **22,** 702–719 (2010).

44. *Scalasca Website* http://scalasca.org/.

45. *Vampir Website* http://vampir.eu/.

46. *Periscope Website* http://periscope.in.tum.de/.

47. *TAU Website* http://tau.uoregon.edu/.

48. Shende, S. S. & Malony, A. D. The TAU parallel performance system. *International Journal of High Performance Computing Applications* **20,** 287–311 (2006).

49. *HPC Toolit Website* http://hpctoolkit.org.

50. Adhianto, L. *et al.* HPCTOOLKIT: tools for performance analysis of optimized parallel programs. *Concurrency and Computation: Practice and Experience* **22,** 685–701. ISSN: 1532-0634 (2010).

51. *ARM MAP Website* https://www.arm.com/products/development-tools/hpc-tools/cross-platform/forge/map.

52. *Intel® VTune™ Amplifier Website* https://software.intel.com/en-us/intel-vtune-amplifier-xe.

53. Shoukourian, H., Wilde, T., Auweter, A., Bode, A. & Piochacz, P. Towards a unified energy efficiency evaluation toolset: an approach and its implementation at Leibniz Supercomputing Centre (LRZ). *on Information and Communication Technologies,* 276 (2013).

54. Kluge, M., Hackenberg, D. & Nagel, W. E. Collecting distributed performance data with dataheap: Generating and exploiting a holistic system view. *Procedia Computer Science* **9,** 1969–1978 (2012).

55. Guillen, C., Hesse, W. & Brehm, M. in *Euro-Par 2014: Parallel Processing Workshops: Euro-Par 2014 International Workshops, Porto, Portugal, August 25-26, 2014, Revised Selected Papers, Part II* (eds Lopes, L. *et al.*) 363–374 (Springer International Publishing, Cham, 2014). ISBN: 978-3-319-14313-2. doi:10.1007/978-3-319-14313-2_31. https://doi.org/10.1007/978-3-319-14313-2_31.

56. *Apache Cassandra Website* https://cassandra.apache.org/ (2016).

57. *DataStax Inc. Website* https://datastax.com/ (2014).

58. Chang, F. *et al.* Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)* **26,** 4 (2008).

59. *Apache HBase Website* https://hbase.apache.org/ (2016).

60. *Apache Thrift Website* https://thrift.apache.org/ (2016).

61. *Cassandra Query Language Reference version 3.3* http://docs.datastax.com/en/cql/3.3/cql/cqlIntro.html (2016).

62. *MQTT Website* https://mqtt.org/ (2014).

63. Banks, A. & Gupta, R. *MQTT Version 3.1.1* tech. rep. (OASIS Standard., Oct. 2014).

64. Intel Corporation, Hewlett-Packard Company, NEC Corporation, Dell Inc. *Intelligent Platform Management Interface Specification Second Generation v2.0 Errata 7* Apr. 2015.

65. *IPMI Adopters List* https://www.intel.com/content/www/us/en/servers/ipmi/ipmi-adopters-list.html (2014).

66. *OpenIPMI Website* http://openipmi.sourceforge.net/ (2006).

67. *Net-SNMP Website* http://www.net-snmp.org/ (2013).

68. ISO. *ISO 8601:2004. Data elements and interchange formats — Information interchange — Representation of dates and times* 29. https://www.iso.org/standard/40874.html (ISO, 2004).

69. *ICT - Information and Communications Technologies — Updated Work Programme 2011 and Work Programme 2012* http://cordis.europa.eu/fp7/ict/docs/3_2012_wp_cooperation_update_2011_wp_ict_en.pdf.

70. Rajovic, N. *et al. The Mont-Blanc prototype: an alternative approach for HPC systems* in *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for* (2016), 444–455.

71. Eicker, N., Lippert, T., Moschny, T. & Suarez, E. The DEEP Project An alternative approach to heterogeneous cluster-computing in the many-core era. *Concurrency and Computation: Practice and Experience* **28.** cpe.3562, 2394–2411. ISSN: 1532-0634 (2016).

72. ISO. *ISO 16484-5:2017-12 Building automation and control systems (BACS) — Part 5: Data communication protocol* 1312. https://www.iso.org/standard/71935.html (ISO, 2017).

73. *American Society of Heating, Refrigerating and Air-Conditioning Engineers Website* https://www.ashrae.org/.

74. Joseph, E. C. *Worldwide HPC Server 2016–2021 Forecast* Market Analysis (Hyperion Research, 2017).