Development of a Power Consumption Dashboard for On-Premises Servers of Small and Medium-Sized Enterprises

HARALD BEIER and PATRICK PRUGGER, University of Applied Sciences Burgenland, Austria

ABSTRACT Rising energy costs pose significant challenges for small and medium-sized enterprises (SMEs) operating on-premise server infrastructures. This paper presents a scalable, serverless energy monitoring system that integrates Internet of Things (IoT) power monitoring, smart meter data, and real-time electricity market pricing. Built on Amazon Web Services (AWS), the system combines data from Sagemcom smart meters and NOUS A5T PowerCable devices with European Power Exchange (EPEX) Spot market prices to enable cost-aware workload management. Comprehensive energy profiling across diverse operational scenarios demonstrates that intelligent workload scheduling based on electricity pricing can reduce operational costs by up to 10.50 cent/kWh in our test intervall for the CPU stress test (peak load). A 24-hour evaluation period revealed distinct power consumption patterns, ranging from 172.2W (idle) to 662.25W (peak load). A practical implementation framework and empirical evidence supporting the economic viability of energy-aware server management for SMEs are provided.

1 Introduction

The rapid escalation of energy prices has emerged as a significant operational challenge for small and medium-sized enterprises (SMEs), particularly those maintaining on-premise server infrastructures. As electricity costs become an increasingly substantial component of IT expenditures, optimizing energy efficiency and cost management is critical for ensuring business sustainability and competitiveness [4]. Server operations, which are indispensable for business continuity, exhibit highly variable energy consumption depending on workload intensity, maintenance activities, and system states.

Despite the importance of energy-aware management, most SMEs lack transparency regarding the power consumption profiles of their server activities. Routine tasks such as reboots, backups, software updates, and maintenance can each have distinct energy footprints, yet their impact on overall operational costs often remains opaque. This lack of actionable insight hinders the ability of SMEs to strategically schedule energy-intensive operations, especially in the context of dynamic electricity pricing [16].

A parallel can be drawn to the management of photovoltaic (PV) systems, where owners are advised to operate high-consumption appliances during periods of peak solar generation. Similarly, SMEs could benefit from aligning server workloads with periods of lower electricity prices or increased renewable energy availability, thereby reducing costs and supporting sustainability objectives.

The detailed system architecture and methodological approach are described in Section 3. Empirical results and analysis are presented in Section 4.

This gap is addressed by proposing a comprehensive, serverless architecture that integrates smart meter data, Internet of Things (IoT)-based server power monitoring, and real-time electricity market pricing into a unified, cloud-based dashboard.

The primary objectives of this research are as follows:

- To develop a scalable system for collecting and analyzing detailed energy consumption data from SME server infrastructures
- To enable real-time correlation of server activities with electricity market prices to inform cost-effective workload scheduling

Authors' Contact Information: Harald Beier; Patrick Prugger, University of Applied Sciences Burgenland, Eisenstadt, Austria.

- To demonstrate the practical benefits of the proposed approach through empirical evaluation across diverse operational scenarios

To achieve these aims, Amazon Web Services (AWS) serverless technologies, including API Gateway, IoT Core, Lambda, and DynamoDB, are leveraged to collect and process data from Sagemcom smart meters and NOUS A5T PowerCable devices. Visualization and analysis are facilitated through Amazon QuickSight, while real-time electricity prices from the European Power Exchange (EPEX) Spot market are integrated via the smartENERGY API (see Appendix B). The complete implementation, including source code and deployment instructions, is made available in a public repository (see Appendix E).

The remainder of this paper is organized as follows: Section 2 reviews related work in energy management and optimization, Section 3 details the system architecture and methodology, Section 4 presents our experimental findings and cost analysis, Section 5 concludes with future research directions, and Section 5 provides references. Detailed API specifications and implementation guides are provided in the appendices.

2 Related Work

A growing body of research addresses the challenges of energy measurement, management, and optimization in server and data center environments. This section reviews key contributions in four thematic areas: (1) process-level energy measurement, (2) data center management approaches, (3) IoT and building management systems, and (4) policy context and foundations. A synthesis of the research gap and the unique contribution of this work is provided. The proposed solution is detailed in Section 3, and a comparison of results with prior work is discussed in Section 4.

Process-Level Energy Measurement: Recent advances have enabled more granular attribution of energy consumption within server environments. Qiao et al. (2023) introduced Wattmeter, a Linux-level tool that attributes energy usage to individual processes and informs CPU scheduling decisions. While this approach is valuable for operating system research, it does not integrate electricity pricing or provide visualization capabilities, limiting its applicability for cost-aware management [15]. Similarly, Zhang et al. (2022) developed a method for container-level energy attribution in microservice architectures, but did not address real-time market price integration. Khan and Varghese (2021) proposed energy-aware scheduling algorithms for heterogeneous server clusters, focusing on power consumption patterns without considering dynamic electricity pricing [8, 22].

Data Center Management Approaches: Energy optimization in data centers has been approached from multiple perspectives. Chen et al. (2022) applied machine learning to enable cost-aware demand response, achieving significant peak-shaving in large-scale cloud environments. Cheung et al. (2021) proposed linear models correlating CPU utilization with power consumption, offering a simpler alternative for data center operators. Smith et al. (2023) analyzed the trade-offs between cost and carbon emissions, highlighting the complexity of multi-objective optimization. Mahmoud et al. (2020) combined workload prediction with renewable energy forecasts for hybrid optimization, while Wang et al. (2023) employed reinforcement learning to dynamically allocate resources in response to variable electricity prices [2, 3, 12, 18, 20].

IoT and Building Management Systems: The integration of IoT technologies has expanded the scope of energy management beyond traditional data centers. Gholami et al. (2020) provided a comprehensive survey of smart building energy management systems, emphasizing IoT architectures and analytics. Ristić et al. (2021) designed an IoT-based energy management platform specifically for SMEs, demonstrating the potential for tailored solutions in smaller

organizations. Palacios-Garcia et al. (2022) explored edge computing for real-time energy management in distributed environments, while Kumar et al. (2021) introduced blockchain-enabled frameworks for secure energy data sharing. Lopez-Garcia et al. (2023) proposed standardized APIs to enhance interoperability among energy management systems [5, 10, 11, 14, 16].

Policy Context and Foundations: Policy and regulatory frameworks play a critical role in shaping energy management practices. The International Energy Agency (2023) provides high-level policy guidance for data centers, and the European Commission (2022) has compiled case studies highlighting successful energy efficiency initiatives in SMEs. Siano (2014) established foundational concepts for demand-response programs, while Ahmad et al. (2021) analyzed regulatory frameworks affecting SME energy management across jurisdictions. The Green Grid Association (2021) introduced advanced metrics for assessing data center sustainability, and Koomey and Masanet (2021) provided updated global estimates of data center energy consumption, underscoring the importance of server-level optimizations [1, 4, 6, 7, 9, 17].

Gap Analysis and Contribution: Despite significant progress in each of these domains, existing solutions typically address only isolated aspects of the broader challenge. To the best of our knowledge, no prior work offers a holistic system that combines per-process energy attribution in SME servers, smart meter data integration, real-time electricity price correlation, and a unified dashboard for actionable insights. This gap is addressed by proposing an AWS-based architecture (Section 3.2) that unifies kernel-level process metering, smart meter feeds (Appendix A), and market price polling (Appendix B) into a comprehensive visualization and decision-support platform. The complete implementation details and source code are available in a public repository (Appendix E), enabling reproducibility and further research in this domain.

3 Methodological Approach

This section outlines the methodological approach taken to design, implement, and evaluate an energy-aware server management solution for small and medium-sized enterprises (SMEs). It describes the overall use case, the technical architecture of the prototype, and the integration of real-time energy consumption and electricity price data.

3.1 Use Case Description

The use case centers on an SME operating on-premise server infrastructure, seeking to optimize energy consumption and costs amid rising electricity prices. Key stakeholders include the SME IT administrator, who manages server operations and energy monitoring, and external data providers such as smart meters and electricity market APIs. The system aims to enable the administrator to schedule energy-intensive tasks during periods of lower electricity prices, thereby minimizing costs and supporting sustainability goals. As illustrated in Figure 1, energy consumption data is continuously collected from both facility-level smart meters and device-level IoT power monitors. Simultaneously, real-time electricity prices are retrieved from market data providers (e.g., EPEX Spot). These data streams are integrated and visualized via a cloud-based dashboard, providing actionable insights and recommendations for workload scheduling.

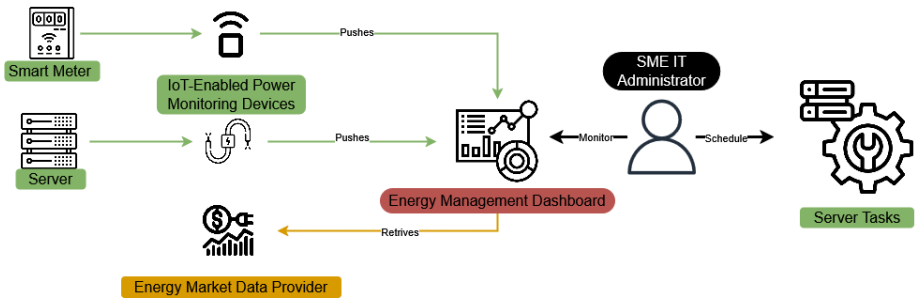

Fig. 1. Overall use case of server administrator utilizing the energy management dashboard.

3.2 Prototype

The prototype implements a cloud-native architecture leveraging Amazon Web Services (AWS) to ensure scalability, reliability, and cost-effectiveness—key considerations for SMEs with limited IT resources. Figure 2 illustrates the system architecture. The complete implementation details, including infrastructure as code and configuration files, are available in the project materials and are further described in Appendix E. At the edge, a smart meter with a P1 interface and a

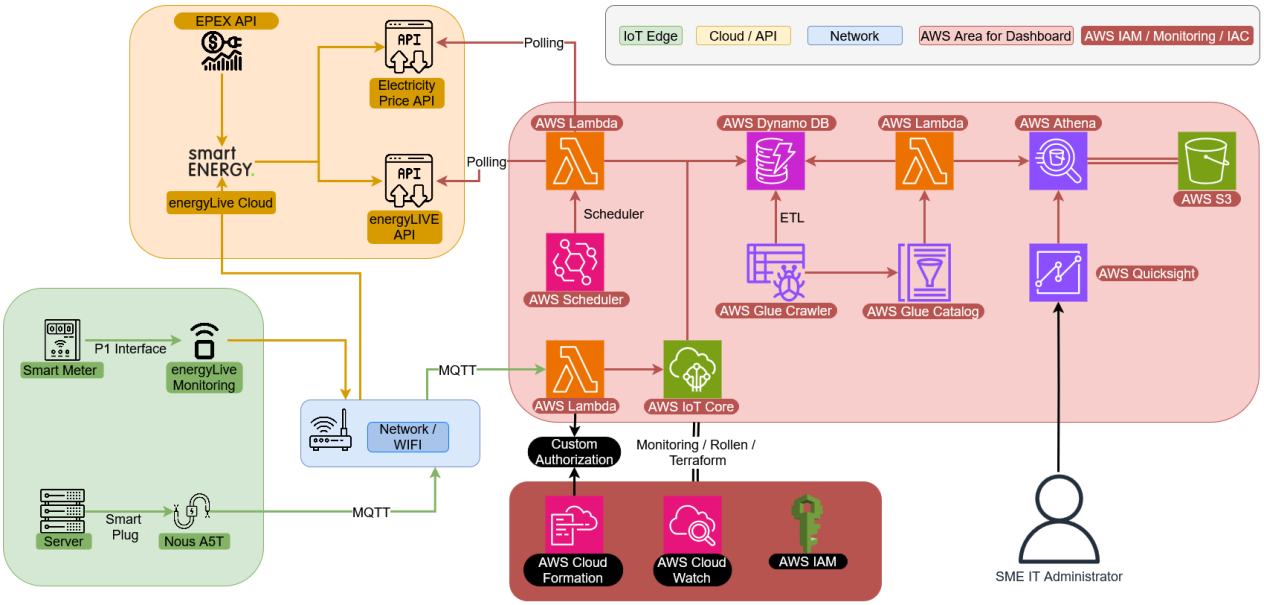

Fig. 2. System Architecture of the AWS implementation.

NOUS A5T smart plug[1] are deployed within the SME's local network. The smart meter provides aggregate facility-level data, while the NOUS A5T delivers device-specific power usage via MQTT. Data is ingested into the cloud using a hybrid of polling (for smart meter and market prices) and message-based (for IoT devices) mechanisms. AWS Lambda functions, triggered by scheduled polling or MQTT messages, process incoming data. AWS IoT Core manages secure

[1]See Appendix C for configuration details.

MQTT communication, while API Gateway and Lambda handle RESTful interactions with external APIs. All data is stored in AWS DynamoDB for low-latency access. AWS Glue and Athena enable automated data cataloging and ad hoc querying, respectively. Visualization is provided through AWS QuickSight dashboards, allowing administrators to correlate energy usage with market prices and identify optimization opportunities. Infrastructure is provisioned using AWS CloudFormation for reproducibility, and security is enforced via AWS IAM and CloudWatch.

3.3 Evaluation Approach

To evaluate the effectiveness of the approach, a comprehensive evaluation strategy is implemented that addresses technical performance, server energy consumption patterns, and economic impacts. This multi-faceted evaluation provides insights into both the system's technical capabilities and its practical utility for SME operators. The results of these experiments are presented in Section 4.

Server energy consumption analysis forms the core of the measurement approach. Using the NOUS A5T PowerCable device, detailed measurements of a single server under various operational scenarios are conducted to establish energy consumption profiles. Specifically, the following workloads (WL) are examined:

- WL1 **Maximum computational load:** Simulation of 100% CPU utilization across all cores using the stress-ng tool on a Linux virtual machine. The stress-ng command will be configured to spawn CPU-intensive worker processes equal to the number of available virtual CPUs, ensuring maximum load across all cores. This scenario represents peak computational demand typical of batch processing or intensive data analysis tasks. [19]
- WL2 **I/O stress testing:** This WL conduct I/O stress testing to evaluate power consumption during intensive disk operations. Using fio (Flexible I/O Tester), we will simulate maximum Solid State Drive (SSD) utilization with sequential and random read/write patterns. This I/O-intensive scenario represents workloads common in database operations, log processing, and large file transfers, providing insights into storage subsystem energy requirements under heavy load.
- WL3 **System reboot cycle:** Measurement of the overall power consumption profile during a full reboot sequence of the host machine, capturing the energy requirements during shutdown, boot, and system initialization phases. This provides insights into the energy costs associated with maintenance operations and system updates.
- WL4 **Maintenance operations:** Monitoring of energy consumption during typical maintenance activities, specifically the patching process of a Linux virtual machine. This scenario represents regular administrative tasks that SMEs must perform to maintain security and system integrity.
- WL5 **Idle state:** Establishing the baseline energy consumption when the server is in an idle state with minimal active processes. This measurement is crucial for understanding the fixed energy costs of maintaining server availability even during periods of low utilization. [13, 21]

For each WL, the total energy usage consumption is measured in kilowatt-hours (kWh) and power consumption patterns over time, establishing detailed energy profiles that can be correlated with specific operational states. These measurements reveal the energy intensity of different server activities and identify potential optimization opportunities, such as scheduling high-consumption tasks during periods of lower electricity pricing or implementing more efficient idle-state management.

To ensure reliable and representative measurements, each test scenario will be conducted with specific time windows (TW):

TW1 For maximum computational load and I/O stress testing scenarios, each test window will span 15 minutes of load and 15 minutes of idle to capture steady-state behavior and account for any thermal effects or performance throttling that may occur during sustained high-load operations.

TW2 System reboot cycle measurements will be conducted over multiple iterations, with each complete cycle (shutdown to fully operational) typically lasting 5-10 minutes.

TW3 Maintenance operation measurements will cover the entire duration of typical update processes, estimated at 5-10 minutes per session, including download, installation, and post-update system stabilization periods.

TW4 Idle state measurements will be conducted over longer 60-minute windows during off-peak hours to establish accurate baseline consumption patterns and capture any periodic background system activities.

A minimum cool-down period of 10 minutes has to be assured between test iterations to ensure thermal conditions return to baseline.

3.4 Test Infrastructure Setup

To conduct the stress testing scenarios described in WL1 and WL2, a dedicated test infrastructure in the form of a Ubuntu 24.04.2 LTS virtual machine was established within a Proxmox virtualization environment.

3.4.1 CPU Stress Testing Configuration. For maximum computational load testing (WL1), a dedicated stress-ng scheduling script was developed to generate sustained CPU load across all available cores. The stress-ng tool was selected for its ability to create reproducible, high-intensity computational workloads suitable for energy consumption analysis.

The CPU stress testing implementation (see Appendix E for implementation details) includes the following characteristics:

- **Full CPU utilization:** Spawns worker processes equal to the number of available virtual CPUs (56 logical processors)
- **Sustained load patterns:** Maintains 100% CPU utilization for the entire 15-minute test duration
- **Thermal consideration:** Includes cooling periods between test cycles to prevent thermal throttling effects
- **Automated scheduling:** Uses the `at` command for precise timing synchronization with energy measurement intervals
- **Comprehensive logging:** Records CPU utilization metrics and timestamps for correlation with power consumption data

The stress-ng configuration utilizes the command `stress-ng --cpu 56 --timeout 900s` to ensure maximum computational load across all processor cores. This approach simulates intensive batch processing workloads typical in SME environments, such as data analysis, compilation tasks, or scientific computing operations.

Similar to the I/O stress testing, the CPU stress script supports configurable test cycles with alternating 15-minute stress periods and 15-minute idle periods, enabling direct comparison of energy consumption between high-load and baseline states. The automated cleanup and logging mechanisms ensure consistent test conditions and comprehensive data collection for subsequent analysis.

3.4.2 Storage Configuration. The original virtual machine configuration had insufficient free disk space for intensive I/O testing that requires substantial temporary file creation. To address this limitation, a dedicated 50GB virtual disk was provisioned and attached to the test virtual machine as a secondary storage device.

The additional disk was formatted with the ext4 filesystem providing 50GB of available space exclusively for I/O testing operations. The disk was configured with I/O threading enabled in the Proxmox hypervisor to maximize I/O performance and ensure realistic energy consumption measurements under high-load conditions. This configuration ensures that:

- I/O testing operations do not interfere with system operations on the primary disk
- Sufficient space is available for creating large test files (up to 32GB total)
- Test data can be isolated and cleaned up after each measurement cycle
- Storage performance characteristics can be measured independently

3.4.3 Automated Test Scheduling. To ensure precise timing alignment with the 15-minute measurement intervals of the energy monitoring system, automated test scheduling scripts were developed using the `at` command scheduler. The FIO stress testing implementation (see Appendix E for implementation details) includes the following characteristics:

- **Configurable test cycles:** Support for multiple 15-minute I/O stress periods followed by 15-minute idle periods
- **Maximum I/O load generation:** Utilizes 4 parallel FIO jobs with 32-depth I/O queues, mixed random read-/write patterns (70% read, 30% write), and variable block sizes (4KB to 1MB)
- **Direct I/O operations:** Bypasses operating system caches to ensure actual disk I/O and realistic power consumption
- **Automatic cleanup:** Removes test files after each cycle to prevent disk space exhaustion
- **Comprehensive logging:** Records detailed performance metrics and timestamps for correlation with energy measurements

This automated approach ensures that I/O stress testing can be precisely synchronized with energy measurement intervals, enabling accurate correlation between storage workload intensity and power consumption patterns.

4 Results

This section presents the key findings from our energy-aware server management system evaluation, focusing on system implementation outcomes and quantitative energy consumption analysis. The complete analysis scripts and raw data are available in our public repository (see Appendix E).

The methodology used to obtain these results is described in Section 3. These findings support the conclusions drawn in Section 5.

4.1 System Implementation

The prototype system was successfully deployed on AWS infrastructure using Infrastructure as Code principles, with all configuration files and deployment scripts available in the repository (see Appendix E). Key technical achievements include:

- Real-time data ingestion with sub-second latency from NOUS A5T smart plugs via MQTT
- Reliable 1-minute interval data collection from energyLive API (see Appendix A)
- Reliable 15-minute interval data collection from EPEX Spot API (see Appendix B)
- Scalable DynamoDB storage with consistent sub-10ms query response times

4.2 Energy Consumption Analysis

Our analysis revealed distinct power consumption patterns across different workload scenarios, as illustrated in Figure 3.

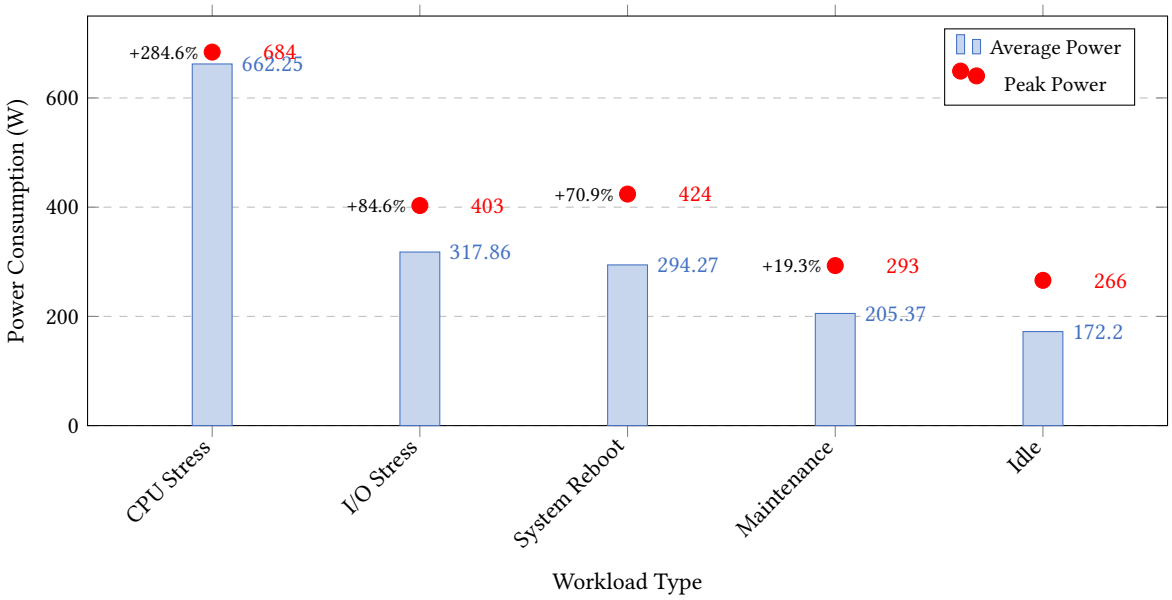

Fig. 3. Comprehensive workload power analysis showing average consumption (bars), peak values (red dots), and percentage increase over idle baseline. The baseline idle power of 172.2W demonstrates significant power variation across different operational states.

Table 1. Workload Power Consumption Details

Workload	Avg (W)	Peak (W)	Var (W)	kWh
CPU Stress	662.25	684.0	±3.94	0.662
I/O Stress	317.86	403.0	±52.15	0.159
Sys Reboot	294.27	424.0	±71.68	0.025
Maintenance	205.37	293.0	±42.59	0.017
Idle State	172.20	266.0	±6.82	0.172

4.3 Cost Optimization Potential

Analysis of EPEX spot prices during the test period revealed significant optimization opportunities:

- Price variation range: -1.685 to 14.168 cent/kWh (230.6% daily variation from reference price)
- Average reference price: 6.874 cent/kWh
- Data points for EPEX spot price: 96 (15-minute intervals)

Table 2 shows, for each workload, the hypothetical cost if all energy had been purchased at the minimum, average, or maximum EPEX spot price observed during the test day. The "cost difference" column quantifies the absolute cost reduction possible for each workload by shifting all consumption from the most expensive to the cheapest period. Negative costs at the minimum price reflect periods of negative electricity pricing, not considering energy grid fees.

Table 2. Workload Cost at Minimum, Average, and Maximum EPEX Spot Price

Workload	kWh	Min Cost (¢)	Avg Cost (¢)	Max Cost (¢)	Cost Difference (¢)
CPU Stress Test	0.662	-1.12	4.55	9.38	10.50
I/O Stress Test	0.159	-0.27	1.09	2.25	2.52
System Reboot	0.025	-0.04	0.17	0.35	0.39
Maintenance Operations	0.017	-0.03	0.12	0.24	0.27

These results demonstrate that intelligent workload scheduling based on real-time electricity pricing can lead to substantial cost savings for SMEs operating on-premise server infrastructure. The combination of comprehensive energy monitoring and price-aware scheduling provides a practical approach to optimizing operational costs while maintaining service quality.

4.4 Implementation Considerations and Limitations

The analysis revealed several key considerations for implementing energy-aware server management:

- **Workload Constraints:**
  - Critical operations cannot always be scheduled optimally
  - Some workloads require immediate execution regardless of energy costs
  - Batch processing jobs offer the most flexibility for optimization
- **Economic Factors:**
  - Price volatility (230.6% daily variation from reference price) enables significant optimization
  - Benefits scale linearly with server count in SME deployments
  - Austrian market prices may not reflect other regional patterns
- **Technical Limitations:**
  - Single server measurements may not represent diverse hardware configurations
  - Virtualization overhead affects absolute power measurements
  - Limited test duration (24 hours) may miss longer-term patterns

5 Conclusion and Future Work

This paper addressed the critical challenge of energy management for SMEs operating on-premise server infrastructure through a cloud-based monitoring and optimization solution. The implementation, detailed in Section 3.2 and Appendix E, successfully integrated IoT-based power monitoring, smart meter data, and real-time electricity pricing to provide actionable insights for cost-effective server operations.

The key contributions of this work include:

- A scalable, serverless architecture combining IoT monitoring and real-time pricing data, as demonstrated in Section 3.2
- Empirical evidence from Section 4.2 demonstrating potential cost savings of up to 10.5 cent per kWh for peak loads (CPU stress test) through intelligent workload scheduling
- A practical framework for SMEs to optimize server operations based on energy costs, with implementation details provided in Appendix E

A summary of the experimental results can be found in Section 4.

While the evaluation demonstrated significant potential for cost optimization (see Section 4.3), several limitations should be noted:

- Single server configuration testing, as described in Section 3.4
- Limited measurement period detailed in Section 4.4
- Focus on Austrian electricity markets (see Appendix B)
- Manual intervention requirements for workload scheduling

Future work should explore:

- Machine learning for automated workload scheduling, building on the energy analysis framework (Section 4.2)
- Multi-server environment support, extending beyond the current setup (Section 3.4)
- Integration with renewable energy sources, complementing the price-based optimization (Section 4.3)

The results establish that energy-aware server management can provide substantial benefits for SMEs, creating a foundation for both cost reduction and environmental responsibility. As energy prices continue to rise, such integrated approaches will become increasingly critical for SME competitiveness. The complete implementation and analysis tools are available in the public repository (Appendix E), enabling further research and practical applications in this domain.

References

[1] T. Ahmad et al. 2021. Regulatory frameworks for energy efficiency in SMEs: A comparative analysis. *Energy Policy* 156 (2021). doi:10.1016/j.enpol.2021.112425

[2] L. Chen et al. 2022. Learning a Data Center Model for Efficient Demand Response. In *Proceedings of the 13th ACM International Conference on Future Energy Systems (e-Energy)*. https://energy.acm.org/eir/learning-a-data-center-model-for-efficient-demand-response/

[3] K. Cheung et al. 2021. Simple Utilization-Based Power Modeling for Data Centers. *Energy and Buildings* 253 (2021). doi:10.1016/j.enbuild.2021.111507

[4] European Commission. 2022. Energy Efficiency in SMEs: Success Stories and Good Practices. https://ec.europa.eu/energy/topics/energy-efficiency/energy-efficient-buildings/energy-efficiency-smes_en

[5] M. Gholami et al. 2020. A Survey on Energy Management in Smart Buildings: Issues and Solutions. *Energies* 13, 13 (2020). doi:10.3390/en13133295

[6] Green Grid Association. 2021. Beyond PUE: Tackling IT's wasted terawatts. https://www.thegreengrid.org/en/resources/library-and-tools/white-papers

[7] International Energy Agency. 2023. Data Centres and Data Transmission Networks. https://www.iea.org/reports/data-centres-and-data-transmission-networks

[8] H. Khan and B. Varghese. 2021. Energy-aware scheduling policies for heterogeneous server clusters. *IEEE Transactions on Parallel and Distributed Systems* 32, 9 (2021). doi:10.1109/TPDS.2021.3079827

[9] J. Koomey and E. Masanet. 2021. Does not compute: Avoiding pitfalls assessing the Internet's energy and carbon impacts. *Joule* 5, 7 (2021), 1625–1628. doi:10.1016/j.joule.2021.05.019

[10] R. Kumar et al. 2021. Blockchain-enabled secure energy trading framework in smart buildings. *IEEE Internet of Things Journal* 8, 10 (2021). doi:10.1109/JIOT.2020.3042528

[11] M. Lopez-Garcia et al. 2023. Standardized APIs for energy management system integration. *Energy Informatics* 6, 1 (2023), 1–18. doi:10.1186/s42162-023-00237-6

[12] S. Mahmoud et al. 2020. Cost-aware workload scheduling of a green data center with prediction and guarantee of renewable energy utilization. *IEEE Transactions on Industrial Informatics* 16, 7 (2020). doi:10.1109/TII.2019.2945377

[13] Marina Moran. 2024. Dissecting the software-based measurement of CPU energy consumption. *arXiv preprint arXiv:2401.15985* (2024). https://arxiv.org/pdf/2401.15985.pdf

[14] E. Palacios-Garcia et al. 2022. Edge computing for real-time energy management: A review. *Applied Energy* 307 (2022). doi:10.1016/j.apenergy.2021.118141

[15] Y. Qiao et al. 2023. Wattmeter: Per-Process Energy Measurement and Scheduling in Linux. In *Proceedings of the 14th ACM International Conference on Future Energy Systems (e-Energy)*. https://energy.acm.org/eir/energy-aware-process-scheduling-in-linux/

[16] B. Ristic et al. 2021. IoT-Based Energy Management Platform for SMEs: Design and Implementation. *IEEE Access* 9 (2021). doi:10.1109/ACCESS.2021.3108721

[17] P. Siano. 2014. Demand Response and Smart Grids-A Survey. *Renewable and Sustainable Energy Reviews* 30 (2014). doi:10.1016/j.rser.2013.10.022

[18] J. Smith et al. 2023. The War of the Efficiencies: Understanding the Tension Between Carbon and Energy Optimization. *ACM e-Energy* (2023). https://energy.acm.org/eir/the-war-of-the-efficiencies-understanding-the-tension-between-carbon-and-energy-optimization/

[19] Ubuntu. 2020. *stress-ng - a tool to load and stress a computer system.* https://manpages.ubuntu.com/manpages/focal/man1/stress-ng.1.html

[20] S. Wang et al. 2023. Reinforcement learning for dynamic resource allocation under variable electricity pricing. *IEEE Transactions on Cloud Computing* 11, 2 (2023). doi:10.1109/TCC.2021.3110644

[21] Jawad Haj Yahya, Haris Volos, Davide B. Bartolini, Georgia Antoniou, Jeremie S. Kim, Zhe Wang, Kleovoulos Kalaitzidis, Tom Rollet, Zhirui Chen, Ye Geng, and Onur Mutlu andYiannakis Sazeides. 2022. AgileWatts: An Energy-Efficient CPU Core Idle-State Architecture for Latency-Sensitive Server Applications. *arXiv preprint arXiv:2203.02550* (2022). https://arxiv.org/abs/2203.02550

[22] K. Zhang et al. 2022. Container-level energy attribution for microservice architectures. *IEEE Transactions on Services Computing* 15, 3 (2022). doi:10.1109/TSC.2020.3042526

A energyLIVE API Specification

The energyLIVE API, provided by Energie Steiermark, enables the integration of real-time smart meter data into third-party systems, thereby supporting advanced energy management and automation solutions. This interface is designed to facilitate the retrieval of consumption data and system status from smart meters, and can be combined with the smartENERGY electricity price API for dynamic tariff applications such as smartCONTROL.

A.1 Authentication and Access

Access to the energyLIVE API requires authentication via an HTTPS header, specifically the `X-API-KEY`, which must contain a valid key obtained from the customer portal. All requests must be made over secure HTTPS connections to ensure data privacy and integrity.

A.2 Base URL and Endpoints

The base URL for the API is:

https://backend.energylive.e-steiermark.com/api/v1/

To retrieve the latest measurements from a specific smart meter interface, the following endpoint is used:

`devices/I-XXXXXXXX-XXXXXXXX/measurements/latest`

where the interface UID is provided by the customer portal.

A.3 Data Format and Response Structure

API responses are returned in JSON format, consisting of an array of measurement objects. Each object contains the following fields:

- **measurement**: Specifies the type of value, typically indicated by an OBIS code for electrical quantities.
- **timestamp**: The time at which the measurement was recorded in the energyLIVE database, represented as a 13-digit Unix timestamp (milliseconds).
- **value**: The measured value, with units such as watt-hours (Wh) for meter readings and watts (W) for instantaneous power.

A.4 Example Request and Response

A typical request to the API using `curl` is as follows:

```
curl -X GET -H "X-API-KEY: <your_api_key>" \
"https://backend.energylive.e-steiermark.com/api/v1/devices/I-10082023-01658401/measurements/latest"
```

The response is a JSON array, for example:

```
[
  {
    "measurement": "0100010700",
    "timestamp": 1726559995000,
    "value": 138.0
  },
  {
```

```
    "measurement": "0100010800",
    "timestamp": 1726559995000,
    "value": 9577201.0
  }
  // ...
]
```

A.5 Use Cases

The energyLIVE API is suitable for a variety of applications, including the integration of smart meter data into home automation platforms, the development of custom energy monitoring dashboards, and the implementation of dynamic energy management strategies based on real-time consumption and pricing data.

For further details and best practice examples, refer to the official documentation and user community resources provided by Energie Steiermark[2].

[2]https://www.smartenergy.at/api-schnittstelle-energylive

B smartENERGY Strompreis API Specification

The smartENERGY Strompreis API, provided by Energie Steiermark, enables customers to access quarter-hourly electricity price data in real time, facilitating the integration of dynamic pricing information into custom systems and applications. This API supports the automation of energy consumption and the optimization of energy management strategies by providing timely and accurate market price data from the EPEX Spot AT electricity exchange.

B.1 Authentication and Access

The Strompreis API is publicly accessible and does not require authentication or special parameters for basic price retrieval. All requests are made via HTTPS to ensure secure data transmission.

B.2 Base URL and Endpoint

The base URL for the API is:

https://apis.smartenergy.at/market/v1/price

A simple GET request to this endpoint returns the latest available electricity price data.

B.3 Data Format and Response Structure

The API returns data in JSON format, with the following structure:

- **tariff**: The tariff identifier, typically set to `EPEXSPOTAT`.
- **unit**: The unit of the price value, e.g., `ct/kWh`.
- **interval**: The validity interval of each price entry in minutes (usually 15).
- **data**: An array of objects, each containing:
  - **date**: The local date and time from which the price is valid.
  - **value**: The price including 20% VAT, in decimal format.

B.4 Example Request and Response

A typical request to the API is as follows:

```
GET https://apis.smartenergy.at/market/v1/price
```

The response is a JSON object, for example:

```
{
  "tariff": "EPEXSPOTAT",
  "unit": "ct/kWh",
  "interval": 15,
  "data": [
    {
      "date": "2023-06-23T00:00:00+02:00",
      "value": 12.592
    },
    ...
  ]
```

```
}
```

B.5 Use Cases

The Strompreis API is particularly useful for automating energy consumption in response to real-time price signals, enabling the development of energy management systems that optimize usage patterns according to periods of low electricity prices. It also supports detailed energy monitoring and cost analysis by providing granular, up-to-date market price data.

For further details and best practice examples, refer to the official documentation and user community resources provided by Energie Steiermark[3].

[3] https://www.smartenergy.at/api-schnittstellen

C Tasmota Configuration for NOUS A5T Power Strip

This appendix details the custom Tasmota configuration used for the NOUS A5T power strip, enabling secure integration with AWS IoT and advanced energy monitoring capabilities. The configuration is designed to facilitate reproducibility and ease of deployment for research and practical applications in energy-aware server management.

C.1 Hardware Overview

The NOUS A5T power strip features four individually controllable AC outlets, three USB ports, and integrated energy monitoring for voltage, current, and power. The device is based on the ESP8285 microcontroller, which provides 1MB of flash memory and supports custom firmware such as Tasmota.

C.2 Configuration Purpose and Features

The custom configuration enables the following key features:

- Secure AWS IoT integration with TLS/SSL support
- MQTT communication for real-time telemetry
- Energy monitoring (voltage, current, power)
- Custom device naming and topic structure
- Web interface and HTTP API for local management
- Rules engine for automation

C.3 Device Template

The following Tasmota template is used to define the hardware configuration for the NOUS A5T:

```
{
  "NAME": "NOUS A5T",
  "GPIO": [0, 3072, 544, 3104, 0, 259, 0, 0, 225, 226, 224, 0, 35, 4704],
  "FLAG": 1,
  "BASE": 18
}
```

C.4 Custom Firmware Build and Flashing

To deploy the custom configuration, the following steps should be followed:

(1) Copy the provided `user_config_override.h` file to the Tasmota source directory:

```
copy tasmota-config\user_config_override.h tasmota\tasmota\user_config_override.h
```

(2) Build the custom firmware using PlatformIO:

```
cd tasmota
pio run -e tasmota
```

(3) Flash the firmware to the NOUS A5T device. The built binary is located at `tasmota/build_output/firmware/tasmota.bin`. Flashing can be performed via the Tasmota web UI or a serial connection.

C.5 AWS IoT and MQTT Setup

After flashing, configure the device for AWS IoT integration using the following Tasmota console command (replace placeholders with your actual AWS IoT endpoint and credentials):

```
BackLog SetOption3 1; SetOption103 1; MqttHost your-endpoint.iot.region.amazonaws.com; MqttPort 443;
    MqttUser tasmota?x-amz-customauthorizer-name=TasmotaAuth; MqttPassword your-password
```

C.6 Device Information

- **Name**: Server PowerMeter
- **Topic**: serverpowermeter
- **Firmware**: Custom Tasmota v14.6.0 with AWS IoT support

For further details, refer to the configuration file `user_config_override.h` and the project README. This setup enables secure, scalable, and reproducible energy monitoring for research and operational deployments.

D DynamoDB Cleanup Script: deleteTimestamps.py

This appendix documents the Python script `deleteTimestamps.py`, which is used to clean up the DynamoDB `SensorData` table by deleting items with timestamps that do not include microseconds. This operation is necessary after migrating to a new timestamp format to ensure data consistency and prevent legacy entries from affecting subsequent analyses.

D.1 Purpose and Functionality

The script scans the `SensorData` table for items whose timestamp attribute (used as the sort key) is exactly 19 characters long, indicating the absence of microseconds. It then interactively deletes these items, using both the partition key and sort key for precise targeting. The script employs exponential backoff to handle DynamoDB throughput limits and provides progress updates during both scanning and deletion phases.

D.2 Prerequisites

- Python 3.x
- `boto3` and `botocore` libraries installed (`pip install boto3 botocore`)
- AWS credentials configured with permissions to read and delete from the `SensorData` table

D.3 Usage Instructions

(1) Ensure your AWS credentials are set up (e.g., via `aws configure` or environment variables).
(2) Run the script in a terminal:

```
python deleteTimestamps.py
```

(3) Review the summary of items to be deleted and confirm when prompted.
(4) The script will delete the identified items, providing progress updates.

D.4 Script Listing

The full source code is provided below for reproducibility:

```
# This script is used to delete items from the SensorData table that have a timestamp
# without microseconds.
# It is used to clean up the table after the migration to the new format.

import boto3
import time
from botocore.exceptions import ClientError

# Initialize DynamoDB
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('SensorData')

print("Checking table schema...")
```

```
# Get table description to find primary key
table_description = table.meta.client.describe_table(TableName='SensorData')
key_schema = table_description['Table']['KeySchema']

print("Table key schema:")
for key in key_schema:
    print(f"  {key['AttributeName']} ({key['KeyType']})")

# Find the partition key and sort key
partition_key_name = None
sort_key_name = None
for key in key_schema:
    if key['KeyType'] == 'HASH': # Partition key
        partition_key_name = key['AttributeName']
    elif key['KeyType'] == 'RANGE': # Sort key
        sort_key_name = key['AttributeName']

if not partition_key_name or not sort_key_name:
    print("ERROR: Could not find partition key or sort key")
    exit(1)

print(f"Partition key: {partition_key_name}")
print(f"Sort key: {sort_key_name}")

print("Scanning DynamoDB table...")

# Count items without microseconds
count_without_microseconds = 0
items_to_delete = []
total_scanned = 0

def scan_with_retry(exclusive_start_key=None):
    """Scan with exponential backoff for throughput exceptions"""
    max_retries = 5
    base_delay = 1

    for attempt in range(max_retries):
        try:
            if exclusive_start_key:
                response = table.scan(
                    ExclusiveStartKey=exclusive_start_key,
```

```
                ProjectionExpression='#pk, #sk',
                ExpressionAttributeNames={
                    '#pk': partition_key_name,
                    '#sk': sort_key_name
                }
            )
        else:
            response = table.scan(
                ProjectionExpression='#pk, #sk',
                ExpressionAttributeNames={
                    '#pk': partition_key_name,
                    '#sk': sort_key_name
                }
            )
        return response
    except ClientError as e:
        if e.response['Error']['Code'] == 'ProvisionedThroughputExceededException':
            if attempt < max_retries - 1:
                delay = base_delay * (2 ** attempt) # Exponential backoff
                print(f"Throughput exceeded, waiting {delay} seconds...")
                time.sleep(delay)
            else:
                raise
        else:
            raise

# Scan with pagination to get all items
try:
    response = scan_with_retry()

    # Process first batch
    for item in response['Items']:
        total_scanned += 1
        timestamp = item[sort_key_name] # Use sort key name
        if len(timestamp) == 19: # No microseconds
            count_without_microseconds += 1
            # Use both partition key and sort key for deletion
            delete_key = {
                partition_key_name: item[partition_key_name],
                sort_key_name: item[sort_key_name]
            }
```

```
            items_to_delete.append(delete_key)

    # Continue scanning while there are more items
    while 'LastEvaluatedKey' in response:
        print(f"Scanned {total_scanned} items so far...")
        time.sleep(0.1) # Small delay between scans

        response = scan_with_retry(response['LastEvaluatedKey'])

        for item in response['Items']:
            total_scanned += 1
            timestamp = item[sort_key_name] # Use sort key name
            if len(timestamp) == 19: # No microseconds
                count_without_microseconds += 1
                # Use both partition key and sort key for deletion
                delete_key = {
                    partition_key_name: item[partition_key_name],
                    sort_key_name: item[sort_key_name]
                }
                items_to_delete.append(delete_key)

except ClientError as e:
    if e.response['Error']['Code'] == 'ProvisionedThroughputExceededException':
        print("ERROR: DynamoDB throughput exceeded. Consider:")
        print("1. Increasing provisioned throughput")
        print("2. Running this script during off-peak hours")
        print("3. Using on-demand billing mode")
    else:
        print(f"ERROR: {e.response['Error']['Message']}")
    exit(1)

print(f"Total items scanned: {total_scanned}")
print(f"Found {count_without_microseconds} items without microseconds")

if count_without_microseconds > 0:
    # Show first few items to delete
    print("First 5 items to delete:")
    for item in items_to_delete[:5]:
        print(f"  {item}")

    # Ask for confirmation
```

```
    confirm = input("Do you want to delete these items? (yes/no): ")
    if confirm.lower() == 'yes':
        # Delete items with rate limiting
        deleted_count = 0
        for item in items_to_delete:
            try:
                table.delete_item(Key=item)
                deleted_count += 1
                if deleted_count % 50 == 0: # Progress update every 50 items
                    print(f"Deleted {deleted_count} items...")
                    time.sleep(0.1) # Small delay
            except ClientError as e:
                if e.response['Error']['Code'] == 'ProvisionedThroughputExceededException':
                    print(f"Throughput exceeded while deleting. Stopped at {deleted_count} items.")
                    break
                else:
                    print(f"Error deleting item: {e.response['Error']['Message']}")
        print(f"Successfully deleted {deleted_count} items")
    else:
        print("Deletion cancelled")
else:
    print("No items found without microseconds")
```

E GitHub Repository and Documentation

This appendix provides permanent references to the project's GitHub repository and its core documentation. The repository contains all source code, configuration files, and documentation required to replicate the research and implementation described in this paper.

E.1 Repository Information

- **Repository URL**: https://github.com/papers-mcce/energyops
- **Main Branch**: `main`
- **Documentation Branch**: `final-paper`

E.2 Core Documentation Structure

The repository's documentation is organized hierarchically, with the main README.md serving as the entry point. Key documentation components include:

E.2.1 Main Project Documentation. The primary README.md (https://github.com/papers-mcce/energyops/blob/main/README.md) provides:

- Project overview and objectives
- Complete directory structure
- Deployment instructions
- Configuration guides
- Replication procedures

E.2.2 Component-Specific Documentation.

(1) **Terraform Infrastructure**
https://github.com/papers-mcce/energyops/blob/main/Deployment/terraform/README.md
   - AWS resource specifications
   - Deployment procedures
   - Security configurations
   - Infrastructure diagrams

(2) **Energy Analysis Tools**
https://github.com/papers-mcce/energyops/blob/main/Deployment/Energy-Analysis/README_energy_analysis.md
   - Data collection scripts
   - Analysis methodologies
   - Visualization tools
   - Result interpretation guides

(3) **Tasmota Configuration**
https://github.com/papers-mcce/energyops/blob/main/tasmota-config/README.md
   - Device setup instructions
   - Custom firmware configuration
   - AWS IoT integration steps

- Troubleshooting guides

(4) **HTML Presentation**
https://github.com/papers-mcce/energyops/blob/main/html-presentation-project/README.md
- Interactive visualization setup
- Deployment instructions
- Customization options
- Browser compatibility notes

E.3 Cross-References and Dependencies

The documentation maintains internal cross-references using LaTeX labels. Key references include:

- Energy Analysis (Section A)
- Price API Documentation (Section B)
- Device Configuration (Section C)
- Database Management (Section D)

E.4 Version Control and Updates

The documentation is maintained under version control, with updates tracked through Git commits and pull requests. Major changes are documented in the repository's release notes and change logs.

For the most current version of any documentation component, refer to the repository URLs provided above. All links are permanent and reference specific commits to ensure reproducibility of the research implementation.

Reproducibility Note

All scripts, infrastructure code, and setup guides referenced in this paper are included in the project materials to support full reproducibility of the research and experiments described.