MetricQ: A Scalable Infrastructure for Processing High-Resolution Time Series Data

Thomas Ilsche, Daniel Hackenberg, Robert Schöne, Mario Bielert, Franz Höpfner, Wolfgang E. Nagel
Center for Information Services and High Performance Computing (ZIH)
Technische Universität Dresden, 01062 Dresden, Germany
{firstname.lastname}@tu-dresden.de

***Abstract*—In this paper we present MetricQ, a novel infrastructure for collecting, archiving, and analyzing sensor data. Core components of MetricQ are a scalable message broker based on the Advanced Message Queuing Protocol, and a newly developed Hierarchical Timeline Aggregation (HTA) storage concept that is specifically designed for timeseries data. HTA requires moderate data processing during data collection and a storage space overhead of about 10 %, and in turn reduces the complexity of typical timeline request from $\mathcal{O}(N)$ to $\mathcal{O}(1)$. This enables access to very large metric timelines spanning years and billions of data points at a performance level that is sufficient for interactive use cases. In contrast to existing solutions in this domain, no relevant information such as very short peaks in the data is discarded. We demonstrate how we use MetricQ with few metrics at very high update rates, e.g., for energy efficiency research, and for a very large number of metrics at moderate update rates, e.g., monitoring data from the electrical and cooling infrastructure of our data center.**

I. Introduction

Modern data centers are abuzz with sensors producing an abundance of measurement data at an ever-increasing rate. This large amount of metric time series data can overwhelm existing monitoring solutions. A scalable and reliable monitoring infrastructure is indispensable to exploit the full potential of this data. A particular challenge is a responsive, yet correct analysis and visualization of the data collected over months or years of operation.

In this paper, we present MetricQ, a novel infrastructure for collecting and storing high-resolution time series data. We combine established high-available message broker software with efficient storage and analysis components specifically designed for large scale, high-resolution time series data. Our production instance of MetricQ is already being used at our local data center facility providing real-time monitoring, diagnostics, and accounting.

The remainder of the paper is structured as follows: Section II provides a short introduction to the infrastructure. Our storage solution is discussed in Section III. In Section IV, we present our deployment and unique use cases not achievable with other solutions. This is followed by Section V, which introduces and compares related concepts. And finally, we summarize our work in Section VI and give an outlook into future work.

II. Scalable Messaging: Concept and Implementation

In order to tackle the complex communication between different kinds of metric data sources and analysis components, we leverage contemporary message-oriented middleware. This approach has several distinct advantages:

1) *modularity*: Components (agents) operate independently.
2) *transparency*: Consumers can utilize metrics from different kinds of sources with a single interface.
3) *robustness*: The highly available message broker can temporarily buffer messages to avoid data loss.
4) *simplicity*: All components in the system only need to open a connection to the message broker.

A. Concept: Using Message-Brokers for Metric Data

Specifically, we chose the Advanced Message Queuing Protocol (AMQP) v0.9 [1] as implemented by RabbitMQ [2]. In AMQP, software clients called *publishers* send messages to one of several *exchanges*, which are hosted by the AMQP *broker*. The exchange further routes the messages to *queues* (also provided by the broker), where the messages are buffered until a *consumer* (software client) retrieves them.

MetricQ encodes metric data with Protocol Buffers (protobuf) into space-efficient AMQP messages. These data messages contain one or more timestamp-value pairs from exactly one measurement point and are routed using the metric name as topic routing key / queue binding key. Accordingly, consumers can freely choose the set of metrics they will receive. RabbitMQ guarantees the ordering of messages, which serves as an important assumption for further processing.

Requests and responses for persistently stored data also use protobuf-encoded AMQP messages. Further, MetricQ uses JSON-encoded remote procedure call (RPC) messages to facilitate configuration and other metadata management. Using multiple exchanges with flexible routing possibilities allows us to utilize the same messaging infrastructure for the different kinds of functionality (live measurements, persistent requests, management RPCs).

Message Queuing Telemetry Transport (MQTT) [3] is an alternative protocol commonly used in monitoring systems [4], [5]. While it uses publish/subscribe similar to AMQP, MQTT does not support the concepts of exchanges and queues. Therefore, the broker cannot buffer any temporary data, which restricts possible use cases and can lead to data loss.

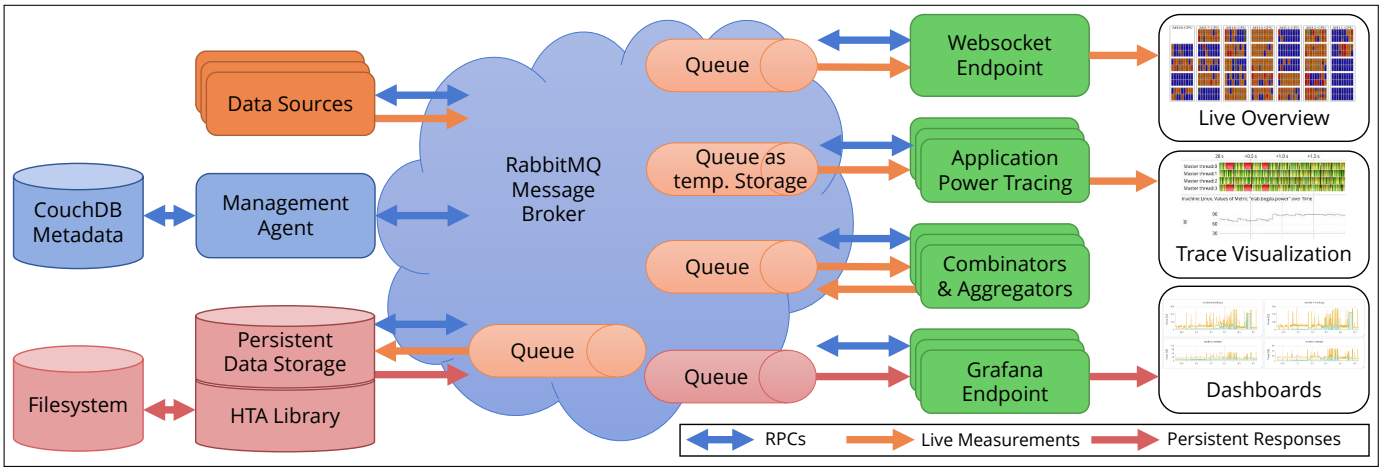


Figure 1: Overview of agents and message-based communication within the MetricQ infrastructure.

B. Implementation: Agents in MetricQ

Figure 1 gives an overview over the types and examples of agents in a MetricQ system. All agents in MetricQ exclusively communicate over AMQP. They only need a RabbitMQ server address and identification token as locally stored configuration. All additional configuration is provided by a MetricQ manager via RPCs. This manager provides access to centralized configuration and metadata access in MetricQ stored in a CouchDB. It also manages the configuration of the message broker, i.e., creating queues and routes, but is never involved in any actual metric data transfer.

The entry points for data in the MetricQ system are data *sources*. They implement a simple interface to publish metric data and the respective metadata to the message broker. Consequently, they are oblivious of how this information is processed any further.

Metric data published to MetricQ is consumed by different kinds of *sinks*. For example, the WebSocket endpoint is a sink which provides live data for web-based dashboards. Queues can also act as temporary storage, e.g., collecting measurement samples for an experiment, which is merged with an application trace in post-processing. While this use case is important for research, it is not in the focus for data center monitoring.

Transformers act as both sinks and data sources and thus allow refinement of metric data. Two specific kinds are currently implemented: On the one hand, *combinators* generate compound metrics as a combination of multiple input metrics, e.g., summaries. On the other hand, *aggregators* can accurately down-sample high-resolution metrics, e.g., for reducing storage requirements.

A special type of sinks is used to *persistently store* metric data. It also provide access to the stored data by answering requests for persistent data. A prominent example that uses such a sink is the Grafana endpoint, which hands data to a Grafana instance, providing versatile charts and dashboards.

The details of the protocol and message encoding is encapsulated by the MetricQ core library. This library allows the rapid development of new agents in C++ and Python.

III. Storage Concept

The large amounts of monitoring data processed by MetricQ present major challenges for the storage back-end. On the one hand, there is a continuously high insertion rate of new data. On the other hand, displaying a long-term timeline chart can require information from millions or even billions of stored measurement samples. Classical timeseries database systems offer two choices: First, traditional aggregate queries can be computed in the database, but require processing of all underlying data, which impairs performance and responsiveness. Second, sampling-based timelines use only a subset of the collected data points, which offers better performance, but represents less information (cf. [6]). However, when presenting a large amount of data in a reduced form for visualization or analysis, it is important to actually retain statistical properties, i.e., minimum, maximum, and mean, as described in the use cases in Section IV. In the following, we describe a concept which exploits the specific properties of metric monitoring data, i.e., append only, monotonic timestamps, and regular insertion rates.

Conventional databases typically use B-trees or Hash-indexes to enable sub-linear seek [7, Chapter 5]. Timeseries database benefit from the order of the collected data and can use Log-Structured Merge-Trees [8, Chapter 4]. However, we chose a flat file and exploit the ordered append-only property of the data. Binary search yields logarithmic complexity without the memory and insertion overhead of a tree. The actual retrieval of data uses a single linear read. Each entry in this file consists of a signed 64 bit POSIX timestamp and a double precision floating point value.

Requests for long-term timeline charts only require statistical aggregations of the raw values, which can be performed in the database server. While this can reduce the network transfer time, the overall cost is still dominated by reading large amounts of data from disk. To overcome this limitation, we define a storage scheme that allows statistical information to be retrieved more efficiently. In addition to the raw values, we store typical aggregation functions, e.g., minimum, maximum, sum, integral,

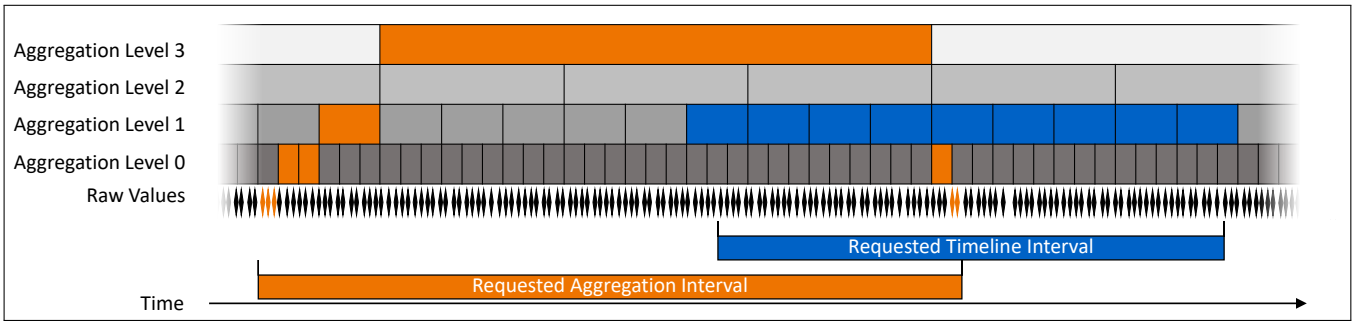


Figure 2: The HTA storage scheme provides precise aggregate values (orange) for arbitrary request intervals and efficient timeline values for a given resolution (blue). In the former case, the algorithm may use all aggregation levels in order to minimize the resource usage to process the query. In the latter case, the choice of aggregation level depends on the number of requested data points, which in typical visualization use cases often depends on the screen resolution.

and count. These can be directly computed from either the raw data or by recursively applying the function on a subset of the data. Therefore, the minimum of any time interval can be computed by using the minimum of the minima of its sub-intervals. This works similarly for other aggregation functions and is done during the insertion of new data.

We now define a time-based profile that aligns with the partitioned time intervals. A single entry in this aggregated timeline holds the statistical information relating to the raw samples within the respective time interval. We further utilize multiple aggregation levels in an approach called *Hierarchical Timeline Aggregation (HTA)*: Based on a number of successive aggregates k in a specific level $l \geq 0$, we generate a new entry at the more coarse-grained timeline level $l + 1$. The lowest aggregation level l_0, which build directly on raw samples, has a constant interval duration of I_0. The interval duration of the different aggregation levels scales with $I_l = I_0 * k^l$. Furthermore, we set a maximum level $l_{\max}$ to avoid small, seldom updated, and almost empty aggregation levels.

Our storage scheme limits the amount of data necessary to compute statistical values over given time intervals. The aligned time intervals have an important advantage: The aggregated values can be indexed with constant effort when stored in simple flat files. In Figure 2, we use four aggregation levels and the raw samples for computing a statistical value for the given time interval. Leveraging the pre-computed aggregations reduces the number of retrieved values from 110 to 10.

The HTA enables efficient statistical timelines as visualized with the second request in Figure 2. Such a request includes a third parameter in addition to start and end time: The minimal required temporal resolution can, for example, correspond to the number of pixels which displays the data. Based on the requested resolution, either a level in the HTA is selected, or the raw samples are used. Hence, the number of accessed data now depends on the resolution. Therefore, the request can be answered in $\mathcal{O}(1)$ with respect to the given time frame. All requests to the flat files of HTA levels use efficient indexing and linear reads. Seeking within the raw sample file is further improved by using the count statistic within the HTA. Computing the offset this way, rather than a binary search, also has $\mathcal{O}(logN)$, but uses more efficient access patterns in practice.

Since the requested resolution doesn't necessarily match an actual aggregation interval, it can be overmatched. This is limited by k for aggregation intervals or the number of samples within I_0 for accessing raw samples. Tuning k and I_0 controls the required storage overhead on the one, and practical performance on the other hand. In practice we use $k = 10$ and $I_0 = 40 \times$ the average sampling rate of the metric. This configuration has a HTA storage overhead of $< 10\,\%$. I_0 is chosen this way, because aggregation records are larger than raw samples ($56\,\mathrm{B}$ vs $16\,\mathrm{B}$).

IV. A Data Center Monitoring Test Case

A. The TU Dresden Data Center and its MetricQ Setup

The main TU Dresden data center LZR was inaugurated in 2015. Designed for up to $5\,\mathrm{MW}$ IT load, it houses all central IT and HPC systems of the TU Dresden on $1200\,\mathrm{m}^2$ of IT floor space. We use MetricQ to collect and store data from a variety of sources, including the BACnet based Building Automation System (BAS) and a large variety of IPMI, SNMP and OpenBMC/HTTP devices (see Table I). Update rates usually range from $1\,\mathrm{Sa/s}$ to $1\,\mathrm{Sa/min}$, with few sources reporting less frequently. The HPC system Taurus features the scalable node-level power measurement solution HDEEM [9],

Table I: Metric Sources used at current LZR installation

Source	Provided Metrics	Aggregated Sample Rate
Power (LMG)*	32	1.21 MSa/s
Power (HDEEM)	4464	4464 Sa/s
PDUs	7308	1736 Sa/s
IPMI	6934	505 Sa/s
BACnet	1792	431 Sa/s
SNMP	1564	307 Sa/s
Combined/Aggregated	599	1087 Sa/s
Others	245	152 Sa/s
Total	22 938	1.221 MSa/s

*:Forwarded to aggregator before stored in HTA

which currently reports three measurements for each of the 1488 compute nodes at 1 Sa/s. For less scalable energy efficiency research at much higher temporal resolution we use several ZES LMG power analyzers with rates up to 151 kSa/s. An aggregator reduces this rate to 100 Sa/s for persistent storage.

Our MetricQ deployment uses two virtual machines, each with 16 hardware threads and 32 GiB memory. The data sources are distributed to measurement-specific systems as they often require specific network access or a physical connection to monitoring devices. Today, a total of 34 agents are continuously connected to the message broker. This includes 23 data sources, three transformers, one aggregator, six HTA storage agents, and the manager. Overall, the system manages ≈23 000 metrics, most of which are recorded in long-term storage. The rate of incoming data is about 1.22 MSa/s. Currently, 13 TiB are used for persistent storage across three NFS-mounted volumes. This includes collected data from up to ten years that was imported from the previous measurement infrastructure.

MetricQ provides a unified interface to this large variety of monitoring data, demonstrating both high cardinality and very high update rates. This enables a range of holistic analysis scenarios from data center monitoring, to HPC system power consumption measurements, to energy-efficiency research.

B. General Data Center Monitoring Use Cases

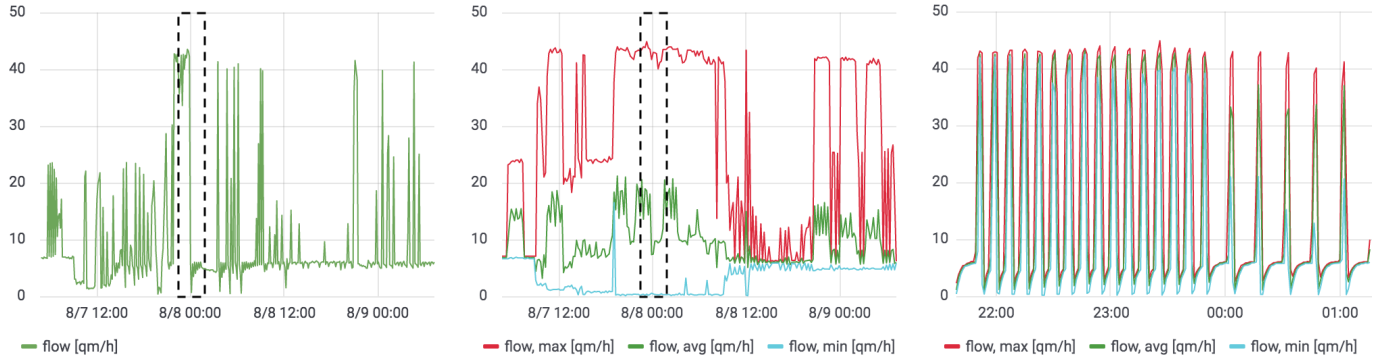
The requirements for a data center monitoring infrastructure include many processes and workflows that are common for most typical IT facilities. This necessitates to support a wide range of data sources across system domains for IT equipment (network, servers, storage, VM hypervisors, batch systems...) and facility equipment (PDUs, cooling towers, chillers, UPS systems, CRAHs, ...). These sources may provide many different measured quantities, e.g., network usage, temperature, power consumption, or HPC node counts. While most of these metrics yield moderate temporal resolution, research projects often require fewer metrics at a much higher sampling rate. All this collected monitoring data needs to be accessed by different stakeholders such as HPC operators, HPC vendors, data center technicians (electrical, cooling), managers, and scientists. Other common tasks based on this data include the creation of regular reports as well as facility planning. A feedback channel from sensors within the IT domain (e.g., the HPC system) into the facility/BAS domain can create an opportunity for advanced system optimization. Some of these use cases have been covered in great detail in other publications, for example [10, Section 4]. With MetricQ, we built a solution to all these cases and additionally support previously impossible usage scenarios.

The regular work of data center technicians includes frequent monitoring of BAS data for irregularities or abnormal behavior. Compared to the standard tools provided by BAS vendors, dashboards (e.g., from Grafana) provide significant advantages: a highly flexible web-based interface that allows technicians to create arbitrary collections of charts in a common view without vendor support. This enables new usage scenarios for all matters that require timeline displays of time series data because live value displays are insufficient. Such dashboards are not only used for day-to-day monitoring of the data center infrastructure, but also to support temporary work, such as installation of new IT resources or changes/extensions of the electrical/cooling support infrastructure.

The usage model for such dashboards is interactive, i.e., scrolling and zooming through arbitrary timelines. This requires short and constant response times [11, Chapter 5], which in turn requires limiting the amount of data that needs to be accessed per request. Section V describes how this is typically achieved through some form of limiting the temporal granularity of the data collection, or accessing only a subset of the available data points for the requested timeframe (sampling). Either way, in order to meet the given performance requirements such compromises are prone to discarding information that is actually required by the user.

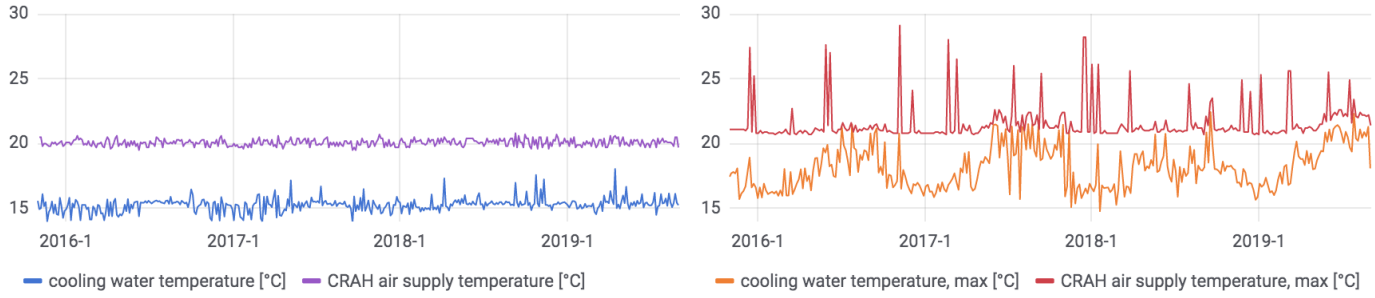
C. MetricQ Advantages for Data Center Monitoring

MetricQ does not require the previously described compromises. The following examples showcase usage scenarios that are highly common in BAS in general and in data center monitoring in particular. Figure 3 shows the volume flow of a badly tuned cooling system during the commissioning phase of a new HPC system. Figure 3a conveys a very different impression than Figure 3b throughout the whole time



(a) classical view (b) MetricQ view (c) MetricQ view, zoomed

Figure 3: Volume flow of a badly tuned cooling system during the commissioning phase of a new HPC system.



(a) classical view

(b) MetricQ maximum view (mean and minimum not depicted)

Figure 4: 46-month data recording of two cooling system temperature probes.

frame. For the highlighted section, chart Figure 3a depicts misleading information about intervals with a steady high or low volume flow, while Figure 3b correctly suggests the fluctuations and even the differing portions of high/low flow depicted in Figure 3c.

While Figure 3b only shows a time frame of about 50 hours, the example depicted in Figure 4 covers a 46-month period. Figure 4a was generated using sampling and therefore hides most of the short temperature fluctuations that usually result from load swings or transitions between free cooling and mechanical chillers. Use cases such as determining the stability of the cooling system or identifying suitable threshold values for alerting purposes require the best- and worst-case values, provided natively by MetricQ as shown in Figure 4b.

The HTA storage concept delivers roughly constant response times, regardless of whether the use case requires looking at small data sets over short time periods, or billions of data points, e.g., multiple metrics at $1\,\mathrm{Sa/s}$ over several years. In data centers, the electrical grid infrastructure is a typical use case that benefits from high sampling rates. Figure 5 shows a MetricQ view that enables a quick assessment of the power quality in an electrical subsystem in terms of voltage fluctuations. Voltages of about $240\,\mathrm{V}$ typically occur during maintenance of the UPS system. In late 2015, a short circuit on a power bus bar occurred while new racks were installed. Exactly one of the >100 million data points ($>1.6\,\mathrm{GB}$) in the depicted time frame covered this incident as voltage drop to $210\,\mathrm{V}$.

MetricQ enables this kind of analysis through a previously impossible combination of nearly constant response times, and retaining and displaying the full level of detail within the data.

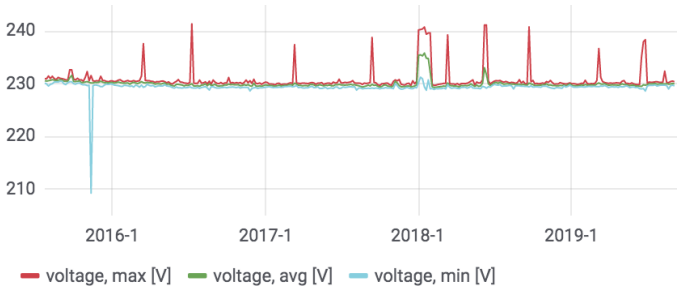


Figure 5: MetricQ view of voltage fluctuations.

V. Related Work

Our previous data center monitoring solution was Dataheap [12]. Its monolithic service for connecting data sources and consumers decreases reliability, and the MariaDB storage agent limits performance, leading to the development of MetricQ.

A. Timeseries Storage

RRDtool [13] is an established tool-set for performance data logging and graphing. It aggregates multiple data points into circular buffers with a fixed number of entries. This limits the necessary storage but discards information that would be relevant to precisely answer aggregate queries.

InfluxDB [14] supports downsampling with custom retention policies. However, these have to be configured manually for each aggregation level and aggregate queries must specifically address a retention policy or aggregate view rather than benefiting from them transparently. TimescaleDB [15] describes continuous aggregates to improve performance of queries on long periods of time, but suffers from similar constraints.

BTrDB [16] is used for synchrophasor measurements and supports out-of-order delivery and compression. It employs a time-partitioned copy-on-write tree, which stores aggregates in intermediate nodes. This approach enables efficient aggregate queries, aggregate insertion rates of up to $53\,\mathrm{MSa/s}$, and statistical queries in under $200\,\mathrm{ms}$. BTrDB lacks functionality for data center monitoring, e.g., centralized configuration of heterogeneous data sources.

B. HPC and Data Center Monitoring

The Lightweight Distributed Metric Service (LDMS) [17] provides monitoring for large scale systems. Its layered infrastructure allows hierarchical scaling. While the overhead is demonstrated for sampling rates up to $1\,\mathrm{Sa/s}$, the two production deployments sample only at $3\,\mathrm{Sa/min}$ and $1\,\mathrm{Sa/min}$ respectively. The deployed data storage uses CSV files.

The Data Center Data Base (DCDB) [5] combines measurements from both facility and compute nodes. MQTT is used for the messaging backend, and Apache Cassandra as storage backend. The authors demonstrate a performance of up to $500\,\mathrm{kSa/s}$ total.

The D.A.V.I.D.E. HPC system uses a framework for fine-grained power and performance monitoring [4]. Embedded computers are attached to each node and publish sensor readings via MQTT. Data is stored in the KairosDB timeseries database and visualized with Grafana. They showcase a total of $47\,\mathrm{kSa/s}$ and individual sensor rates up to $1\,\mathrm{kSa/s}$. With DiG [18], the embedded components can be used for local processing at higher temporal resolution up to $50\,\mathrm{kSa/s}$.

The Operations Monitoring and Notification Infrastructure (OMNI) [10] uses RabbitMQ, Logstash, Elasticsearch, and Grafana to process and visualize metric data from a heterogeneous set of distributed sources. The authors describe that the system can currently ingest up to $25\,\mathrm{kSa/s}$.

Similar goals are addressed by the Grand Unified Information Directory Environment (GUIDE) [19]. GUIDE collects data from storage systems, schedulers, interconnect, and compute nodes of an HPC system and processes it with Splunk.

Overall, none of the existing monitoring solutions are suitable for our use case. While cardinality is generally well-supported, it is not clear whether the existing solutions could handle individual measurements well beyond $1\,\mathrm{Sa/s}$. No full-stack monitoring solution supports efficient aggregate queries for plotting and analyzing years of data at high-resolution. Of the described solutions, DiG offers flexible local analysis and GUIDE supports post-processing and analysis within the unified data streams. Other solutions perform analysis only on stored data. In contrast, MetricQ leverages a single interface to access high-resolution live data, temporarily stored data for experiments, and aggregated persistent measurements. Easily manageable configuration is another challenge not addressed by existing solutions: For most infrastructures, measurement configuration is done locally. DCDB provides remote configuration at the cost of requiring a inbound network connection to all Pushers. Contrary, MetricQ offers a central (re)configuration for components of the system.

VI. Conclusion and future work

In this paper, we presented MetricQ, a modular infrastructure for collecting and storing high-resolution time series data. MetricQ leverages an established message broker to connect data sources, storage, and analysis components. It integrates a novel storage concept that drastically improves the latency for typical metric timeline requests. We demonstrate how we use the MetricQ infrastructure to generate previously unachievable insights into the operation of our data center. By collecting high resolution power measurements up to $151\,\mathrm{kSa/s}$ and storing them at reduced resolution we demonstrate that MetricQ is fit for the increasing fidelity of sensors in data center and elsewhere. All components of MetricQ are provided as open source at https://github.com/metricq.

In future work, we want to measure the performance capabilities of MetricQ in a controlled benchmarking environment. Further, we plan to extend the analysis capabilities, e.g. by offering an integration with Python or R (cf. [20]). We also plan to significantly improve the integration of BACnet-based metrics in terms of automation and administration.

Acknowledgments

This work is supported in part by the German Research Foundation (DFG) within the CRC 912 - HAEC. We thank Maik Schmidt for his tremendous support for the MetricQ project.

References

[1] "Amqp advanced message queuing protocol," AMQP Working Group, Tech. Rep., 2008. [Online]. Available: https://www.rabbitmq.com/resources/specs/amqp0-9-1.pdf

[2] Pivotal Software, Inc., "Rabbitmq documentation," 2019. [Online]. Available: https://www.rabbitmq.com/documentation.html

[3] Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta, "MQTT version 5.0," 2019. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

[4] A. Bartolini, A. Borghesi, A. Libri, F. Beneventi, D. Gregori, S. Tinti, C. Gianfreda, and P. Altoè, "The d.a.v.i.d.e. big-data-powered fine-grain power and performance monitoring support," in *Proceedings of the 15th ACM International Conference on Computing Frontiers*. New York, NY, USA: ACM, 2018, pp. 303–308.

[5] A. Netti, M. Mueller, A. Auweter, C. Guillen, M. Ott, D. Tafani, and M. Schulz, "From facility to application sensor data: Modular, continuous and holistic monitoring with DCDB," *CoRR*, vol. abs/1906.07509, 2019.

[6] S. K. Jensen, T. B. Pedersen, and C. Thomsen, "Time series management systems: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 11, pp. 2581–2600, nov 2017.

[7] B. Schwartz, P. Zaitsev, and V. Tkachenko, *High performance MySQL: optimization, backups, and replication*. " O'Reilly Media, Inc.", 2012.

[8] S. Alapati, *Expert Apache Cassandra Administration*. Apress, 2017.

[9] T. Ilsche, R. Schöne, J. Schuchart, D. Hackenberg, M. Simon, Y. Georgiou, and W. E. Nagel, "Power measurement techniques for energy-efficient computing: Reconciling scalability, resolution, and accuracy," in *SICS Software-Intensive Cyber-Physical Systems*, Apr. 2018. [Online]. Available: https://doi.org/10.1007/s00450-018-0392-9

[10] E. Bautista, M. Romanus, T. Davis, C. Whitney, and T. Kubaska, "Collecting, monitoring, and analyzing facility and systems data at the national energy research scientific computing center," 2019.

[11] J. Nielsen, *Usability Engineering*. London: Academic Press, 1993.

[12] M. Kluge, D. Hackenberg, and W. E. Nagel, "Collecting distributed performance data with dataheap: Generating and exploiting a holistic system view," *Procedia Computer Science*, vol. 9, pp. 1969 – 1978, 2012, proceedings of the International Conference on Computational Science, ICCS 2012.

[13] T. Oetiker, "Rrdtool logging & graphing," 2017. [Online]. Available: https://oss.oetiker.ch/rrdtool/

[14] InfluxData Inc., "InfluxDB 1.X: Open Source Time Series Platform," 2019. [Online]. Available: https://www.influxdata.com/time-series-platform/

[15] Timescale, Inc., "TimescaleDB docs," 2019. [Online]. Available: https://docs.timescale.com

[16] M. P. Andersen and D. E. Culler, "Btrdb: Optimizing storage system design for timeseries processing," in *14th USENIX Conference on File and Storage Technologies (FAST 16)*. Santa Clara, CA: USENIX Association, 2016, pp. 39–52.

[17] J. Brandt, A. Gentile, J. Mayo, P. Pebay, D. Roe, D. Thompson, and M. Wong, "Resource monitoring and management with OVIS to enable HPC in cloud computing environments," in *2009 IEEE International Symposium on Parallel & Distributed Processing*. IEEE, may 2009.

[18] A. Libri, A. Bartolini, and L. Benini, "Dwarf in a Giant: Enabling Scalable, High-Resolution HPC Energy Monitoring for Real-Time Profiling and Analytics," *ArXiv e-prints*, Jun. 2018.

[19] S. S. Vazhkudai, R. Miller, D. Tiwari, C. Zimmer, F. Wang, S. Oral, R. Gunasekaran, and D. Steinert, "GUIDE: a scalable information directory service to collect, federate, and analyze logs for operational insights into a leadership hpc facility," in *SC17: International Conference for High Performance Computing, Networking, Storage and Analysis on*. ACM Press, 2017.

[20] J. Frenzel, Y. Sastri, C. Lehmann, T. Lazariv, R. Jäkel, and W. E. Nagel, "A generalized service infrastructure for data analytics," in *2018 IEEE Fourth International Conference on Big Data Computing Service and Applications (BigDataService)*, March 2018, pp. 25–32.