MonSTer: An Out-of-the-Box Monitoring Tool for High Performance Computing Systems

Jie Li*, Ghazanfar Ali*, Ngan Nguyen*, Jon Hass†, Alan Sill*, Tommy Dang*, and Yong Chen*

* Texas Tech University, Lubbock, TX
Email: {jie.li, ghazanfar.ali, ngan.v.t.nguyen, alan.sill, tommy.dang, yong.chen}@ttu.edu

† Dell EMC, Inc., Austin, TX
Email: jon.hass@dell.com

***Abstract*—Understanding the status of high-performance computing platforms and correlating applications to resource usage provide insight into the interactions among platform components. A lot of efforts have been devoted into developing monitoring solutions; however, a large-scale HPC system usually requires a combination of methods/tools to successfully monitor all metrics, which will lead to a huge effort in configuration and monitoring. Besides, monitoring tools are often left behind in the procurement of large-scale HPC systems. These challenges have motivated the development of a next-generation out-of-the-box monitoring tool that can be easily deployed without losing informative metrics.**

In this work, we introduce *MonSTer*, an "out-of-the-box" monitoring tool for high-performance computing platforms. MonSTer uses the evolving specification *Redfish* to retrieve sensor data from Baseboard Management Controller (BMC), and resource management tools such as Univa Grid Engine (UGE) or Slurm to obtain application information and resource usage data. Additionally, it also uses a time-series database (e.g. InfluxDB) for data storage. MonSTer correlates applications to resource usage and reveals insightful knowledge without having additional overhead on the application and computing nodes. This paper presents the design and implementation of MonSTer, as well as experiences gained through real-world deployment on the 467-node Quanah cluster at Texas Tech University's High Performance Computing Center (HPCC) over the past year.

***Index Terms*—High Performance Computing, monitoring and maintenance tool, visualization tool**

I. Introduction

Monitoring is critical to the successful operation of complex high performance computing (HPC) systems. Failures in HPC not only affect the currently running jobs but also waste scheduling and queuing time. Additionally, failures can result in the loss of data that have already been processed. Therefore, timely monitoring of the status and functionality of the HPC system enables system administrators to adequately handle potential problems to maintain the health of the system and to provide stable computing services for HPC users.

Several different monitoring tools have been proposed over the last decades. Each tool has its advantages and disadvantages. System administrators often select one or several tools to form a dedicated monitoring infrastructure, depending on monitoring needs. To ensure that these tools work as expected, a lot of effort has been put into configuring and customizing them to fit the platform being monitored and meet specific monitoring requirements. Once the HPC infrastructure is revamped, the monitoring tools need to be upgraded, which requires extra effort to implement these changes. Given the complexity of deploying a monitoring platform and timely operation of the monitoring tool, it is desired for a next-generation, out-of-the-box, and scalable monitoring tool.

Maintaining the "keep everything" methods seems to be the safe way to drill down into problems. However, as the HPC systems continue to grow and more metrics can be collected and monitored, moving and saving a greater amount of data than before becomes another challenge. We need to rethink what we care about and keep data that are valuable for detecting anomalies. We can cross-compare and correlate the sub-components within the HPC system, such as jobs data, resources usage and hardware status, so as to quickly understand the system status, detect anomalies in time, and provide guidance for finding and solving problems.

As the Redfish protocol [1] has evolved, many vendors have adopted the Redfish protocol in their server products and expose an interface for accessing BMC information through the Redfish API, which allows out-of-band retrieval of necessary hardware information and sensor data, such as power usage, temperature and fan speed. On the other hand, large-scale production HPC clusters deploy resource managers to schedule applications across computing resources. The resource manager has the overall status of the computing resources and can allocate appropriate resources to jobs submitted in the queue to achieve effective scheduling. The status of computing resources mainly includes CPU usage, memory allocation, and I/O statistics. MonSTer leverages these existing techniques to monitor the system by retrieving metrics from the provided APIs. It correlates collected metrics and reveals insightful knowledge of how platform components interact with each other without having additional overhead on applications and computing nodes.

The following summarizes the contributions of this work:

- We designed an "out-of-the-box" monitoring tool - which we named MonSTer (i.e., Monitoring System Tool) - to collect data from the resource manager and BMCs.

The data reported in this research were collected on Intel and Dell EMC hardware platforms.

- We proposed a middleware in MonSTer that processes and aggregates the collected data. The middleware also provides an API for data analysis.
- We developed a data analysis tool, *HiperJobViz*, which gains insightful information by analyzing and visualizing application data and device status.
- MonSTer largely eliminates the effort to develop customized monitoring tools, allowing system administrators to focus on higher-level tools that leverage the data collected and delve into useful information.

The rest of the paper is as follows. We first describe the background and motivation for developing MonSTer in Section II. In Section III, we then discuss the monitoring architecture and explain the design of each module. Next in Section IV, we present performance overhead and experience gained from a production deployment. Finally, we report related works in Section V and conclude the paper in Section VI.

II. Background and motivation

In this section, we introduce the background of this research including the cluster being monitored, *the Quanah cluster*, and its currently deployed monitoring infrastructures. We also discuss the shortcomings of current solutions and present the motivation for developing the next-generation and "out-of-the-box" monitoring tool.

A. The Quanah Cluster

The Quanah cluster at High Performance Computing Center (HPCC) of Texas Tech University [2] is commissioned in early 2017 and expanded to its current size in 2019, which is comprised of 467 nodes with Intel XEON processors providing 36 cores per node. Quanah has a total of 16,812 cores with a benchmarked total computing power of 485 Teraflops/s and provides 2.5 petabytes storage capability. The cluster is based on Dell EMC PowerEdge™ C6320 servers, which are equipped with the integrated Dell Remote Access Controller (iDRAC) [3] providing Redfish API [1] for accessing Baseboard Management Controller (BMC). The software environment is based on CentOS 7 Linux, provisioned and managed by OpenHPC, and has a fully non-blocking Omni-Path 100 Gbps fabric for MPI communication. The cluster is operated with Univa Grid Engine (UGE) [4], setting up with multiple queues, with jobs sorted by projects to meet the needs of research activities for many fields and disciplines.

B. Existing Monitoring Solutions

The initial motivation for monitoring was the desire to keep track the running status at both hardware and software level. To this end, our production data center, HPCC, deployed an open-source monitoring framework Nagios [5] for HPC infrastructure monitoring. We also used Univa Unisight [4] to monitor and report job scheduler data and Grid Engine environments.

Nagios is an open-source framework for monitoring systems and provides infrastructure failure detection. It alerts administrators and performs automatic recovery mechanisms when problems occur. Recently we developed a Redfish plugin for Nagios [6]. The plugin integrates the Redfish API with Nagios Core, communicates with HPC monitored entities via the Redfish API, and populates node status to Nagios. Additionally, Nagios provides an interactive web interface that allows users to access monitoring details and modify monitoring parameters including removing alerts, executing service checks and more.

Univa Unisight is a robust monitoring and alerting system for managing and analyzing resource usage across a monitored cluster. It enables HPC administrators to control a comprehensive and transparent data collection system and to generate various reports across the cluster. Additionally, the pre-built dashboard allows administrators to easily query and visualize job-related metrics.

There are many other open-source monitoring tools focusing on monitoring high performance computing cluster system. Some tools are only capable of capturing node state which includes CARD [7], Parmon [8], Supermon [9] and Ganglia [10]. A few monitoring tools are capable of job-level monitoring such as Ovis [11] and TACC Stats [12].

C. Motivation

The aforementioned tools used in the Quanah cluster and other existing tools we have surveyed and investigated (discussed in more detail in Section V) do not meet the need for in-depth understanding of systems and applications performance for the following reasons. Nagios is designed for failure detection and notification, and historical monitoring data are collected and saved in text files, through which the accessing and processing are difficult and time-consuming. Furthermore, due to the department policy, the Unisight database is not available to researchers other than HPCC staff. Additionally, none of the existing monitoring tools is capable of analyzing and visualizing user-level information and node-level health status in real-time.

These shortcomings of current solutions have motivated us to build a new monitoring tool on top of the existing infrastructures, which is capable of analyzing systems and user behaviors. The monitoring tool should be available "out-of-the-box" (i.e. it should be easy to deploy, does not require intervention in the running system, and should be scalable to meet further monitoring needs). The tool should be architecture agnostic too so that other HPC systems can adopt it.

III. Architecture

In this section, we first provide a high-level overview description of the MonSTer architecture. We then explain each functional module in detail and discuss the design considerations as well.

A. Overview

A high-level diagram of MonSTer is shown in Figure 1, where solid arrows indicate the data flow, and dashed arrows indicate the instructions flow. As shown in the diagram, there are four main modules:

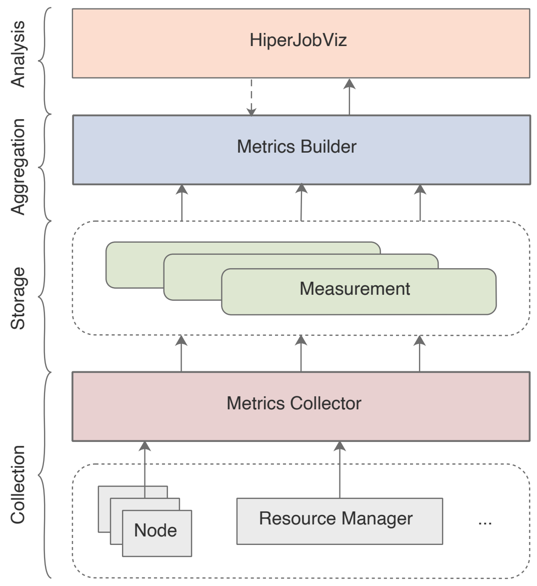


Figure. 1. Diagram of MonSTer architecture (solid arrows indicate data flow; dashed arrows indicate instructions flow.)

1) *Metrics Collector*, which captures interested data from computing nodes and resource managers.
2) A measurement storage module, which consists of a set of measurements for storing the collected data in a time-series database.
3) *Metrics Builder*, which correlates and aggregates data from multiple measurements, and exposes an API to various consumers.
4) A data analysis tool, HiperJobViz, which visualizes and analyzes application data and device status.

HiperJobViz represents a class of analysis tools that can utilize the data provided by Metrics Builder API. We decouple data analysis and data collection to improve the flexibility of the system. Next, we will introduce each module in detail.

B. Collection

HPC monitoring covers a wide range of possible data sources including computing nodes, resource managers, and operating systems. Our monitoring system primarily utilizes out-of-band measurements retrieved via BMCs and in-band measurements accessed through the resource manager. In the current implementation, we do not involve any other collection tools. Figure 2 illustrates the collection components. A centralized collecting agent named Metrics Collector periodically sends requests to the BMCs on computing nodes and UGE on the head node at a pre-defined and configurable collection interval. Metrics Collector then pre-processes, builds time stamps, and writes data points into the storage component.

1) Querying BMC: As we have mentioned in Section II-A, modern computing nodes in HPC systems are equipped with BMCs (such as iDRACs in the Quanah cluster), which provide Redfish API to read system telemetries. Communications between Metrics Collector and BMCs take place over an

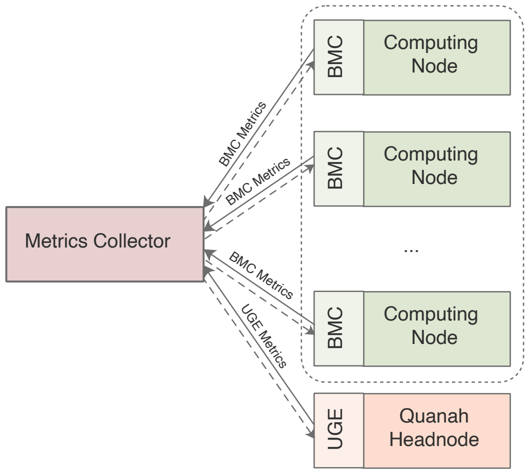


Figure. 2. Diagram of Metrics Collector

independent management network. Therefore, this approach avoids perturbing ongoing computations across compute fabric and allows status to be obtained even if the computing node is down.

More specifically, the current version of iDRAC (model: 13G DCS, firmware version: "2.63.60.61") supports four categories of telemetry data. Selective metrics are listed in table I. Metrics in the same category are exposed as a Redfish API query that can be accessed via a unique URL. For example, to query the **"Thermal"** information from the node **"10.101.1.1"**, the URL is `"https://10.101.1.1/redfish/V1/Chassis/System.Embedded.1/Thermal/"`. At each collection interval, retrieving nine metrics from all 467 nodes in these four categories requires Metrics Collector to build a request pool with 1868 URLs. Metrics Collector sends all requests asynchronously and waits for the responses.

TABLE I
SELECTIVE METRICS COLLECTED FROM BMC

Category	**Metrics**
Manager	BMC Health
System	Host Health
	Processor Health
	Memory Health
Thermal	CPU Temperature
	Inlet Temperature
	Fans Speed (Fan_1, Fan_2, Fan_3, Fan_4)
Power	Power Usage
	Voltages

The current version of iDRAC has limited resources and cannot handle a large number of requests. To improve the success rate of querying, we implement the connection timeout, read timeout, and retry mechanisms in Metrics Collector. In our experience, a Redfish API request takes 4.29 seconds on average. Asynchronous request for all metrics from all nodes takes about 55 seconds.

2) Querying UGE: UGE [4] is a resource management system previously forked from Sun Grid Engine (SGE) [13]. It

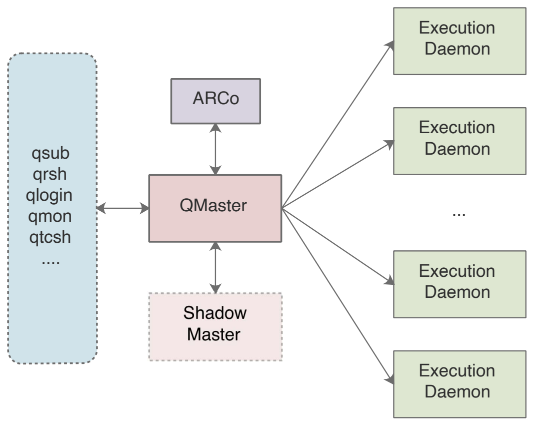


Figure. 3. Univa Grid Engine components architecture

supports and operates in a heterogeneous network environment and is used to manage distributed resources world widely. Figure 3 shows the components of the UGE. Users submit jobs through command-line utilities *qsub*, *qrsh*, etc. The *qmaster* is the core component of UGE that accepts incoming jobs and assigns a holding area where jobs wait to be executed. There may be one or more *shadow masters* to take responsibility in case of failure. The qmaster dispatches jobs with the highest priority when computing resources are available to execute. An *execution daemon* hosted on the available node receives jobs from the qmaster and executes them locally. When a job is completed, the execution daemon notifies the qmaster so that the next job can be scheduled to the empty slot.

Qmaster receives continuous status (such as CPU or memory usage) from execution daemon at fixed intervals, with the default being 40 seconds. If the qmaster fails to receive a continuous status report, the qmaster labels the executing host and its resources as no longer available, and the following jobs are not assigned to this host. While we can set the reporting time to a higher frequency, it may block the qmaster, especially with a high number of execution hosts. Also, frequent load updates are of little value because the load on the execution node usually rises and falls smoothly [14].

As shown in the diagram, UGE also has a component called *ARCo*, short for the Accounting and Reporting Console, which is a web-based tool for accessing accounting information. Metrics Collector uses ARCo to access computing resource metrics and application details. Table II presents a selection of metrics. In addition to collecting resource metrics from UGE, Metrics Collector also supports query metrics from *Slurm*, another widely used resource management and scheduling system.

3) Pre-processing: Metrics Builder extracts metrics from BMCs and UGE. However, these metrics are not passed directly to the storage component. We pre-process metrics in the collection phrase to obtain the following benefits.

Pre-processing the collected metrics has significantly reduced the amount of data. For example, instead of storing all health information, we keep only abnormal status, such as

TABLE II
SELECTIVE METRICS COLLECTED FROM UGE

Category	Metrics
Resource	CPU Usage
	Average Load
	Total Memory
	Used Memory
	Used Swap
	Free Swap
Job	User Name
	Job Name
	Job ID
	Job Submit Time
	Job Start Time
Relationship	Job List on Node

“Warning” or “Error”, to reduce redundancy, as the majority of systems is usually healthy. This optimization reduces a large amount of unnecessary, redundant health data in the system. Additionally, we use binary integers to represent the state, not as strings as in “Warning” or “Error”, which provides a much more compact representation of data collected and monitored. For time stamps, such as job submission time and start time, the data collected from job schedulers are also strings, representing human readable dates and time. We convert these date strings to an integer epoch time, thus achieving significant savings on storage as well. The use of integers and binary data in general, instead of ASCII strings, to represent state and time stamps also makes further analysis easier to implement.

Through pre-processing, more insights can be gained from job-related metrics. For example, based on the “Job List on Node” information, we can summarize how many cores a job uses and how many nodes a job takes up. In addition, since UGE does not provide real-time job finish time, we calculate finish time by comparing the difference between two consecutive job lists. If a job is in the previous list, but not in the current job list, then that job should be completed before the current collection interval. This estimated finish time can be updated when ARCo provides an accurate finish time. Additionally, we calculate memory usage from total memory and used memory to standardize resource usage metrics.

4) Collecting Interval: As discussed in section III-B1 and section III-B2, the collection interval is limited by the response time of the BMC and the job scheduler state update time, which are 55 seconds and 40 seconds on our platform, respectively. Therefore, it is not possible to obtain high-resolution data in seconds or even milliseconds from the existing infrastructures. However, even with high-resolution data, system administrators generally do not respond at a time scale of seconds [15]. Besides, it is challenging to transmit and store high-resolution data points. Therefore, our current implementation keeps the telemetry data collected at a reasonable interval of 60 seconds to ensure that BMC metrics are retrieved even when network fluctuations are encountered and to collect resource manager metrics whenever possible.

C. Storage

In our system, based on the collection interval and the number of metrics obtained from BMCs and UGE, the Quanah cluster generates approximately 1.4×10^7 individual data metrics per day. Traditional SQL databases are inadequate to store and query such large amounts of time series data. On the other hand, it is unacceptable to go through complex queries that can take a lot of time to produce usable results. In order to achieve high usability and scalability from storage module, we choose the open-source time series database InfluxDB [16] as the main storage component and conduct a number of customized optimizations for HPC monitoring tool needs. Moreover, InfluxDB contains a variety of features that can be used to calculate aggregation, roll-ups, downsampling, etc., making the data visualization and data analysis more efficient.

The measurements in InfluxDB can be thought of as SQL tables. We organize them by data sources and category. Currently, we have measurements like *Health*, *Power*, *Thermal*, *UGE*, *JobsInfo*, and *NodeJobs*. The metrics are organized as follows: *Health* measurement stores all health-related information obtained from the BMC including BMC health and system health. *Power* measurement is used to store power usage at the node level and *Thermal* measurement stores CPU temperature, inlet temperature, and fans' speed. CPU usage and memory usage obtained from the UGE are stored in the *UGE* measurement. We use a specialized measurement, *JobsInfo*, to store the details of the job, and *NodeJobs* to store the correlation between the nodes and the job. Sample data points are shown in Figures 4 and 5.

```
"time": 1583792296
"measurement": "Power"
"tags":
    "NodeId": "10.101.1.1"
    "Label": "NodePower"
"fields":
    "Reading": 273.8
```

Figure. 4. Sample data point for storing node power usage of node "10.101.1.1". We add a "Label" in "tags" so that the power consumption of other components can also be saved to the "Power" measurement.

```
"time": 1583892564
"measurement": "NodeJobs"
"tags":
    "NodeId": "10.101.1.1"
"fields":
    "JobList": "['1291784', '1318962',
                '1318307', '1318324']"
```

Figure. 5. Sample data point for storing jobs running on node 10.101.1.1. UGE allows jobs to share the node. Therefore, multiple jobs might be running on the same node at the same time. Data types in InfluxDB do not include array; thus, we stringify the job list information.

Every 60 seconds, Metrics Collector builds data points from the collected metrics based on the schemas. The total number of data points generated within each interval is approximately 10,000, which is the ideal batch size for InfluxDB. Metrics Collector then writes these data points into the database in batches. This approach reduces the network overhead of opening and closing HTTP links by transmitting more data at once.

D. Aggregation

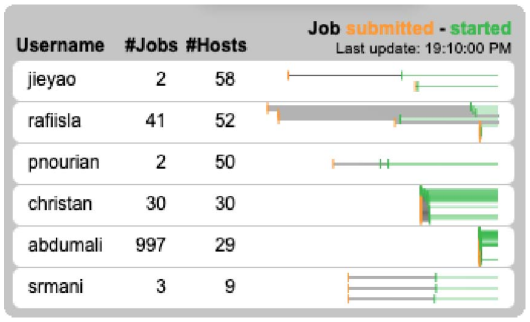
The data stored in the database is raw data, and analysis tools usually digest long-term historical data with different granularities. Querying directly from the database requires an understanding of the precise database schemas and involves additional programming to query and process the data. Additionally, querying and transmitting long-term data requires significant waiting time. To address these issues, we introduce Metrics Builder, an aggregation module in MonSTer. Metrics Builder hides the details of querying InfluxDB, speeds up querying and transmitting, and provides a unified API for data analysis consumers. Decoupling data acquisition and data analysis through a time-series database greatly improves the flexibility and structure of the system.

Metrics Builder acts as a middleware between the consumers (i.e. analytic clients or tools) and the producers (i.e. the databases). Its main workflow is as follows. First, it receives requests from consumers. The request includes time range, time interval and data type information. The time range represents the window of time for the data that consumers want to access. Time intervals and data types are used to aggregate and downsample time series data. Second, Metrics Builder generates the appropriate InfluxDB query strings based on the information specified by the consumer and then sends queries to InfluxDB and waits for responses. Third, Metrics Builder then processes data returned from InfluxDB, builds the data in JSON format, and sends them to the consumer.

We use an example to illustrate some more details. A data analysis tool invokes Metrics Builder API with parameters including a **time range** from **"2020-04-20T12:00:00Z"** to **"2020-04-21T12:00:00Z"** with a **time interval** of **"5m"** and a **data type** of **"max"**. These parameters indicate the data collected between the time window will be downsampled at a maximum of every five minutes. Metrics Builder then generates a series of query strings. For example, to retrieve the power usage of node "10.101.1.1", the following *Influx query language* string will be created:

```
"SELECT max(Reading) FROM Power
 WHERE NodeId='10.101.1.1'
 AND Label='NodePower'
 AND time >='2020-04-20T12:00:00Z'
 AND time < '2020-04-21T12:00:00Z'
 GROUP BY(5m)"
```

Metrics Builder concurrently sends all query requests to InfluxDB. InfluxDB's built-in aggregate function finds the maximum reading of node power of every 5 minutes and returns the aggregated value to Metrics Builder. After all responses are received, Metrics Builder aggregates all metrics by the node ID. Compression is also used to reduce the amount


Job submitted - started
Last update: 19:10:00 PM

Username	#Jobs	#Hosts
jieyao	2	58
rafiisla	41	52
pnourian	2	50
christan	30	30
abdumali	997	29
srmani	3	9

Figure. 6. Timeline visualization of 1-day job scheduling of the Quanah Cluster: Only a few representative users/jobs are shown in the figure.

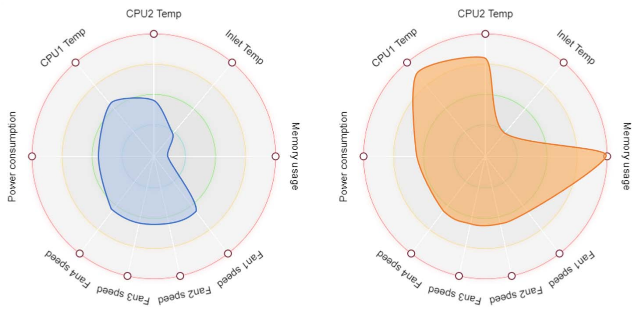


Figure. 7. Radar representations for nine-dimensional metrics of two example computing nodes: (left) Normal status (right) High CPU temperature and high memory usage.

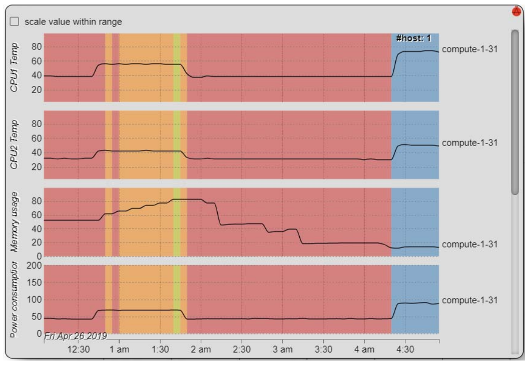


Figure. 8. Historical status change trends for node “1-31”. The background colors highlight the operational states of node “1-31’ over particular time intervals.

of data in order to reduce the transmission time for long-term data. Experimental results on compression performance can be found in Section IV.

E. Data Analysis

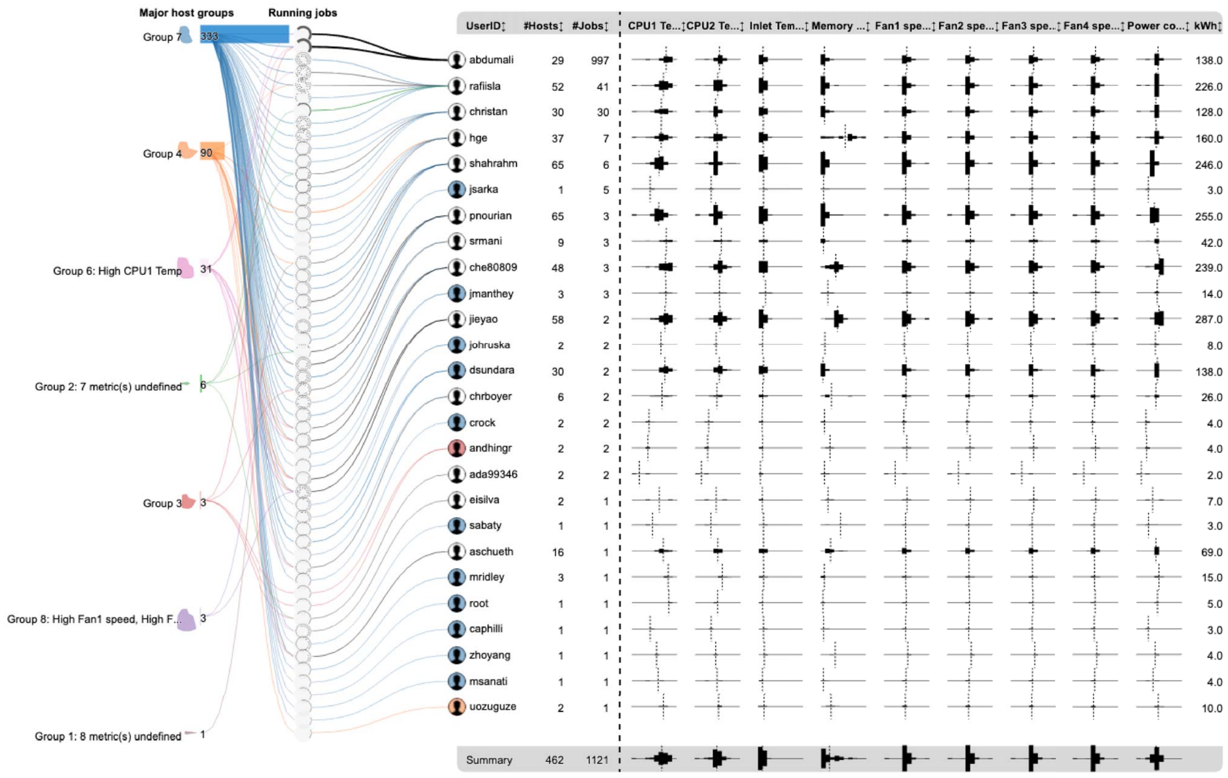
The purpose of the data analysis component is to provide administrators insights of the current status of computing nodes and running jobs, as well as a long-term analysis of historical data using an appropriate graphical representation [17]. For this purpose, we have developed a dedicated data analysis tool, HiperJobViz [18], which invokes Metrics Builder API to retrieve data on demand, tracks changes in resource usage by users and jobs, and visualizes status metrics of computing nodes using high-dimensional representations. A full discussion of HiperJobViz is beyond the scope of this paper. In this paper, we present two main visual components. For those interested in its features, please visit the demo page [19].

1) Job Scheduling Visualization: This component visualizes the status of submitted jobs (an example is shown in Figure 6). The gray bar indicates waiting time before execution, and the green bar indicates running time. From the figure, we can observe that some jobs start executing without any queuing time after submission, while others wait for quite a long time. In addition, the number of jobs submitted by a user and the number of hosts used by the user are expressed in numbers. For example, user “jieyao” submitted two jobs that require 58 hosts at the same time, which indicates the user is running MPI jobs. User “abdumal” submitted 997 jobs, but only occupies 29 hosts. It is likely that these jobs are array jobs that share 36 cores on each host.

Moreover, in order to correlate the job information with the computing nodes, a hierarchical representation is developed, as shown on the left of Figure 9. The running jobs are mapped to these groups, revealing the correlation between the jobs and the status of the computing nodes. The major host groups on the left represent the nodes, which are clustered in seven groups using the k-means clustering algorithm [20]. The blue cluster (*Group 7* on the top left corner) is the most popular cluster, as this is the normal status of the computing nodes (the metric readings are in the middle ranges). We will discuss the node status representation in Section III-E2.

2) Node Health Status Visualization: We use a radar chart to visualize high-dimensional data of each node [21]. As shown in Figure 7, it normalizes the health metrics, organizes them cyclically, and connects relevant values on each dimension to represent the “morphology” of the data profile [22]. Two sample radar charts in this example depict the perceptional differences of two typical statuses in the system: blue is for a normal operational state while orange is for a critical status (high CPU and memory usage). In order to characterize the primary health of the system, we perform a modified k-means clustering of these nine health metrics for the computing nodes. This results in the major host groups that were discussed in Section III-E1.

A matrix of symmetric histograms summarizes the variance of the readings of each user’s data per dimension shown on the right of Figure 9, which provides a visual summary for comparing resource usage across users. By clicking on the attribute name, the histograms will be arranged in ascending or descending order, from which we can easily find the specific user that consumes the most resources. By clicking on the radar chart of each computing node, the historical trend of the metric over time can be shown for further investigation [23]. Figure 8 shows the temperatures, memory usage, and power



UserID	#Hosts	#Jobs	kWh
abdumali	29	997	138.0
rafiisla	52	41	226.0
christan	30	30	128.0
hge	37	7	160.0
shahrahm	65	6	246.0
jsarka	1	5	3.0
pnourian	65	3	255.0
srmani	9	3	42.0
che80809	48	3	239.0
jmanthey	3	3	14.0
jieyao	58	2	287.0
johruska	2	2	8.0
dsundara	30	2	138.0
chrboyer	6	2	26.0
crock	2	2	4.0
andhingr	2	2	4.0
ada99346	2	2	2.0
eisilva	2	1	7.0
sabaty	1	1	3.0
aschueth	16	1	69.0
mridley	3	1	15.0
root	1	1	5.0
caphilli	1	1	3.0
zhoyang	1	1	4.0
msanati	1	1	4.0
uozuguze	2	1	10.0
Summary	462	1121	

Figure. 9. An example snapshot of the Quanah Cluster: Hierarchical structure of job scheduling data and multivariate health status of computing nodes.

consumption of node "1-31" collected between 12 am and 5 pm on April 26, 2019. The colors indicate the clustering group that the status belongs to in a particular time window.

IV. Evaluation and Experiences

In this section, we examine the performance overhead introduced by the monitoring system on a production cluster and the experiences gained by optimizing the monitoring system performance. We have deployed Metrics Collector service, Storage service, and Metrics Builder service on different hosts. Table III describes the hardware specifications of the hosts.

A. Evaluation

Performance overhead mainly consists of two parts: one is intra-node overhead, such as CPU overhead and memory footprint within each computing node; and the other is inter-node overhead, which is the network traffic generated between nodes.

As mentioned in Section III-B, out-of-band measurements avoid interfering with ongoing computation fabric and communicate with Metrics Collector over an independent network, thus querying node status from the BMC has no impact on the computation. Moreover, no extra in-band measurement agents (e.g. monitoring utilities running in an operating system) are used within computing nodes. Therefore, MonSTer does not incur intra-node overhead on the cluster.

TABLE III
Host Hardware Specifications

Metrics Collector Host:	
CPU:	2 x 4 cores Intel Xeon @ 2.53GHz
RAM:	23 GB DDR3
STORAGE:	2TB HDD
NETWORK:	1Gbit/s, Broadcom NetXtreme II
Storage Host:	
CPU:	2 x 8 cores Intel Xeon @ 2.50GHz
RAM:	94 GB DDR3
STORAGE:	400GB SSD, 500GB HDD
NETWORK:	1Gbit/s, Broadcom NetXtreme
Metrics Builder Host:	
CPU:	2 x 8 cores Intel Xeon @ 2.50GHz
RAM:	125 GB DDR3
STORAGE:	24TB HDD
NETWORK:	1Gbit/s, Broadcom NetXtreme

The inter-node overhead caused by MonSTer is querying job scheduler accounting information, which is done only between the host running Metrics Collector service and the cluster headnode running the job scheduler (e.g. UGE qmaster). The network bandwidth traffic is obtained by dividing the number of bytes of accounting data by the UGE query interval (60 seconds). Each node and job information are about 19 KB and 23 KB, respectively. Table IV summarizes the network bandwidth consumed for the transmission of accounting information, which includes network traffic for 467 nodes and an average of 400 jobs information. Our measurements show that the overall bit rate of the monitored data is negligible compared to the speed of the management network, which is usually an Ethernet network.

TABLE IV
NETWORK BANDWIDTH CONSUMED FOR TRANSMISSION OF ACCOUNTING INFORMATION

Monitoring BW	Monitoring BW/Node	Monitoring BW/Job
298.43 KB/s	0.32 KB/s	0.38 KB/s

B. *Experiences and Discussions*

In this sub-section, we present our experiences in optimizing the performance of the monitoring system. In our original design and implementation, even though the data collection module, Metrics Collector, worked well, Metrics Builder typically required an unacceptable waiting time to query and process data from InfluxDB. Figure 10 shows the data query performance at different time intervals over different time ranges. In general, time increases along with the time range at the same time interval. When the query interval is small, time increases quickly. In our experiments, even the shortest time was about 50 seconds, which indicates that Metrics Builder is not a responsive service.

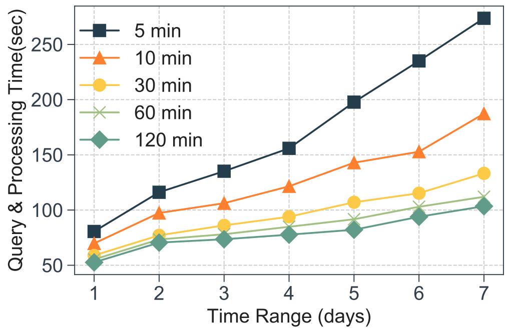


Figure. 10. Query and processing time at different time intervals over different time ranges

To find out what is causing the poor performance, we used *cProfile* to analyze the details of time consumption in Metrics Builder. As shown in Figure 11, querying BMC-related data points takes almost 80% of the total running time, and querying UGE-related data points takes more than 10% of the time. These queries account for about 90% of the total running time. Therefore, if we can save querying time, we can achieve significant performance improvement. To try to save querying time, we have explored the following optimization strategies.

1) Storing data on SSDs: The original InfluxDB service resided on a host where data are stored on HDDs (hard disk drives) with a disk bandwidth of 103MB/sec. To test out the performance of using SSDs (solid state drives) without affecting the collection process, we migrated the collected data to a host with SSDs, which provides around 391 MB/sec of

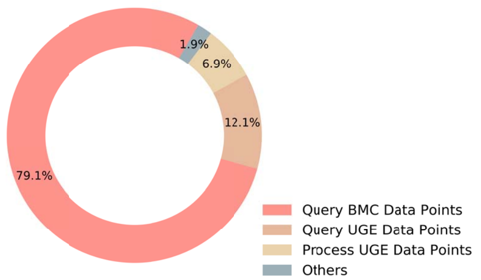


Figure. 11. Time consumption for querying and processing data points from InfluxDB

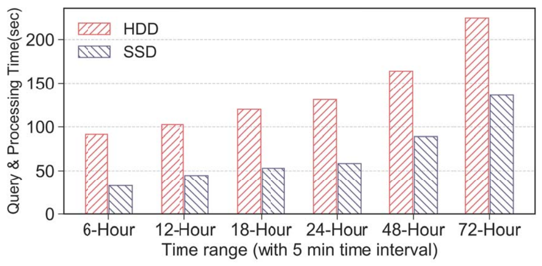


Figure. 12. Query and processing time using HDDs and SSDs

I/O bandwidth, nearly 4x faster than an HDD. As depicted in Figure 12, even if we use faster storage, the performance gains are limited, which is roughly **1.5x** to **2.1x** faster, and the response time is still long.

2) Optimizing database schemas: Our next approach is to redesign the time-series database schema. This optimization is based on the knowledge that query performance scales with series cardinality. In our original design and implementation, we had two versions of the database schema. The first version used different measurements to store different metrics. For example, CPU temperature, fan speed, and job information were all stored into different measurements. We also saved metadata such as threshold information into *fields*. The second version saved all metrics into a unified measurement and each job information is stored into a dedicated measurement. Both versions of the schema coexist in the same database, which introduced a large series of cardinality.

In order to better manage the data, we proposed an optimized schema and converted all historical data into this redesigned schema. We used binary integer epoch time instead of date strings as described in Section III-C. The optimized schema not only results in considerable performance improvement but also reduces the total data volume. As shown in Figure 13, the new schema has only 28.02% of the data volume of the previous schema. We also gained a **1.6x** to **1.76x** performance boost compared to using the previous schema on the SSDs as depicted in Figure 14. These experiences have shown that database schema plays a vital role in the performance of the monitoring system.

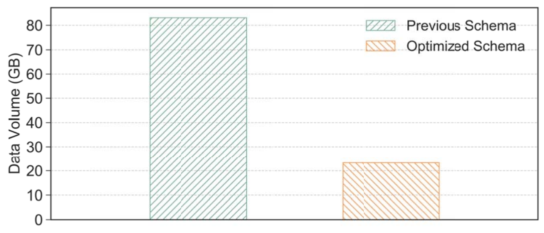


Figure. 13. Data volumes of using the previous schema and the optimized schema. Data were collected between March 14, 2019 and April 10, 2020

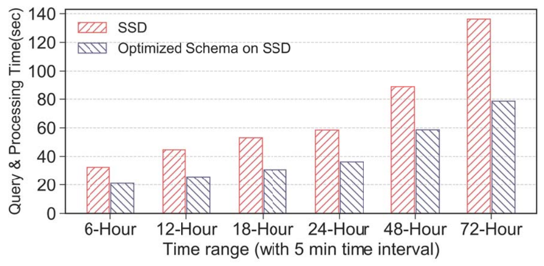


Figure. 14. Query and processing time using SSD and new schema on SSD

3) Concurrent Querying: The next approach we have investigated to improve performance is to take advantage of concurrent queries in InfluxDB, where data points in each measurement are searched in a concurrent manner. Figure 15 shows the performance improvement compared to the sequential approach. We achieved **5.5x** to **6.5x** performance improvement from concurrent querying, which shows that concurrent querying is another vital technique and design consideration in the monitoring system.

Figure 16 summaries the performance improvements using the above approaches collectively. Overall, the proposed strategies allow Metrics Builder to perform **17x** to **25x** faster than the original implementation. The query and processing time was as low as 3.78 seconds when querying 6 hours of data and 12.9 seconds when querying 72 hours of data.

4) Transmitting compressed data: These optimization approaches we have discussed earlier collectively reduce the waiting time to an acceptable range. However, when data

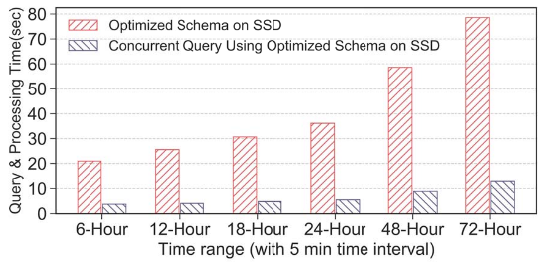


Figure. 15. Query and processing time using new schema on SSD and concurrent query

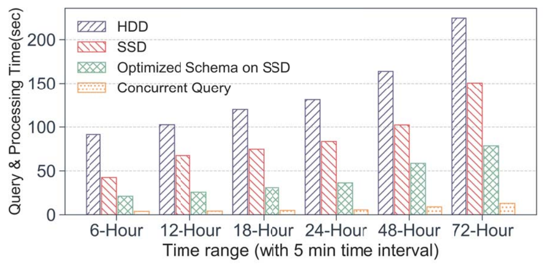


Figure. 16. Performance achievements of different optimizations

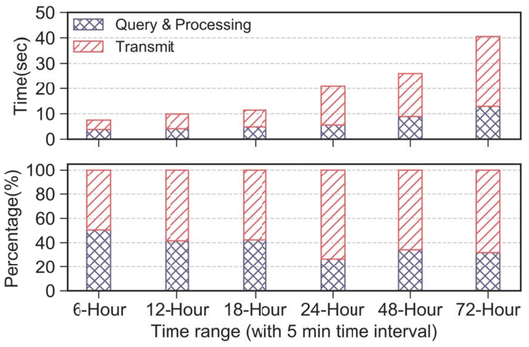


Figure. 17. Query-processing and transmission time (on top) and time distribution (at bottom)

analysis invokes Metrics Builder API remotely, the response time is still long, especially when obtaining long-range data. To further understand the reasons, we decompose the time consumption into query-processing time and transmission time, as depicted in Figure 17. From the figure, we can observe that when querying long-range data, the transmission time is much longer than the query-processing time, up to 1.65 times longer. This observation motivates another optimization of compressing data points and transmitting compressed data to reduce the transmission time.

As discussed in Section III-D, Metrics Builder API provides JSON format data to data analysis. In our experiments, we used *zlib* [24] library to compress JSON data into compressed data format. Figure 18 illustrates the compression ratio. The compressed data volume is only about **5%** of the uncompressed data volume. Figure 19 shows the total response time (on top) and the total response time distribution (at bottom) when using compressed data, compared against the cases without compression. Attributing to the compression, the transmission time is significantly shorter and the overall performance is about **2x** faster than transmitting uncompressed data even though the query-processing time increases only slightly.

V. Related Work

There are a number of open-source and commercial tools focusing on monitoring high performance computing cluster systems. For instance, CARD designed by Anderson et. al. [7]

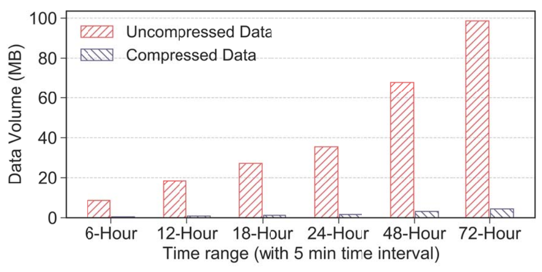


Figure. 18. Data volumes of uncompressed data and compressed data

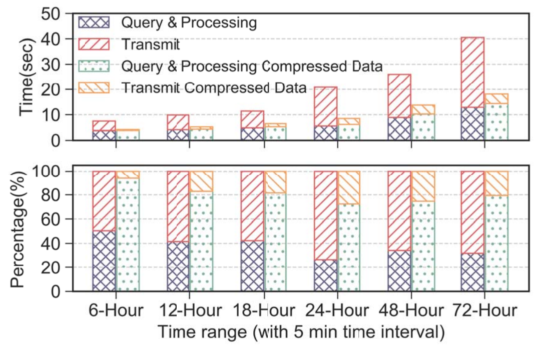


Figure. 19. Query-processing and transmission time for uncompressed and compressed data (on top) and time distribution (at bottom)

is an early exploration of monitoring large-scale clusters. It uses a relational database to store collected data. Parmon [8] is a similar system that collects performance-related and system health data from nodes. Supermon presented by Sottile et. al. [9] is a set of tools for monitoring systems with high speed and low impact. It introduces symbolic expressions to represent data at various levels, making it composable and hierarchical. The Ganglia proposed by Massie et. al. [10] is another scalable distributed monitoring system. It uses a multicast-based listen/announce protocol to monitor states within clusters and aggregates status by creating a tree of point-to-point connections between representative cluster nodes. LDMS [25] allows various metrics such as memory usage, network bandwidth to be collected at a very high frequency.

A few monitoring tools are capable of job-level monitoring. As an example, Ovis [11] collects job data from the scheduler log file and provides statistical insights about applications. Del Vento et. al. [26] proposed a method to associate floating-point counters with jobs and identifies poorly performing jobs. TACC Stats [12] is a component of the HPC systems analytic project, SUPPeMM [27], that runs the data collector module in the job scheduler prolog to collect resource usage data for each job. However, TACC Stats writes the collected data into a text file, which is then aggregated every 24 hours. Therefore, it does not support real-time analysis and visualization of the data.

Summary: MonSTer differs from these tools discovered above in several ways. First, MonSTer uses a hybrid and holistic approach to collect status data of the system without introducing any new collection daemons on the computing nodes. Second, MonSTer only causes a negligible network overhead on the cluster headnode and incurs no overhead on the computing nodes. Third, MonSTer correlates out-of-band measurements and in-band measurements, which allows statistical analysis of resource usage information at the user-level flexibly. Finally, MonSTer provides near real-time analysis and visualization of user-level information and node-level health status.

VI. Conclusions and Future Work

In this paper, we have presented the architecture and design details of an "out-of-the-box" monitoring tool that requires minimal configurations. MonSTer utilizes the Redfish API to retrieve sensor data from the BMC on each computing node and the resource manager (i.e. the job scheduler) for job information and resource usage data. It processes the monitoring metrics and saves data points into a time-series database, InfluxDB. A dedicated analysis and visualization tool consumes data from the provided API and visualizes user-level information and node-level metrics in near real-time, providing insightful knowledge for system administrators and users. Our experiences with the deployment on a production 467-node cluster have shown that well-designed schema can reduce the amount of data collected without losing informative metrics and using high-speed storage, concurrent processing, and transmitting compressed data reduce data retrieval time significantly and enable near real-time analysis and visualization.

On the other hand, MonSTer relies heavily on existing infrastructures and therefore, the metrics collected and the frequency of monitoring are limited by the capabilities of those infrastructures. MonSTer currently does not include file system and network monitoring capabilities yet, and cannot retrieve BMC metrics within seconds. In the near future, we will collect more metrics by using additional tools and the upcoming telemetry model. The data analysis tool can also be upgraded to visualize new metrics.

Metrics Builder API and HiperJobViz are currently open to researchers who are interested in testing and exploring these features. For more information about Metrics Builder API, please visit the webpage [28]. For HiperJobViz, please visit the demo page [19].

VII. Acknowledgement

We are thankful to Dell EMC and Intel for their valuable support of this project through the Cloud and Autonomic Computing (CAC) Center membership at Texas Tech University site. This research is supported in part by the National Science Foundation under grants OAC-1835892, CNS-1817094, and CNS-1939140. We are also very grateful to the High Performance Computing Center of Texas Tech University for providing HPC resources for this project.

References

[1] DMTF. (2020) DMTF's Redfish®. [Online]. Available: https://www.dmtf.org/standards/redfish

[2] HPCC. (2020) High Performance Computing Center. [Online]. Available: http:www.depts.ttu.edu/hpcc/

[3] D. Technologies. (2020) Integrated Dell Remote Access Controller (iDRAC). [Online]. Available: https://www.delltechnologies.com/en-us/solutions/openmanage/idrac.htm

[4] UGE. (2020) Univa Grid Engine. [Online]. Available: https://www.univa.com/

[5] Nagios. (2020) Nagios-The Industry Standard In IT Infrastructure Monitoring. [Online]. Available: https://www.nagios.org/

[6] G. Ali. (2018) Nagios Redfish API Integration: Out-of-band (BMC) based Monitoring. [Online]. Available: https://github.com/nsfcac/Nagios-Redfish-API-Integration

[7] E. Anderson and D. A. Patterson, "Extensible, scalable monitoring for clusters of computers." in *LISA*, vol. 97, 1997, pp. 9–16.

[8] R. Buyya, "Parmon: a portable and scalable monitoring system for clusters," *Software: Practice and Experience*, vol. 30, no. 7, pp. 723–739, 2000.

[9] M. J. Sottile and R. G. Minnich, "Supermon: A high-speed cluster monitoring system," in *Proceedings. IEEE International Conference on Cluster Computing*. IEEE, 2002, pp. 39–46.

[10] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30, no. 7, pp. 817–840, 2004.

[11] J. M. Brandt, B. J. Debusschere, A. C. Gentile, J. R. Mayo, P. P. Pébay, D. Thompson, and M. H. Wong, "Ovis-2: A robust distributed architecture for scalable ras," in *2008 IEEE International Symposium on Parallel and Distributed Processing*. IEEE, 2008, pp. 1–8.

[12] T. Evans, W. L. Barth, J. C. Browne, R. L. DeLeon, T. R. Furlani, S. M. Gallo, M. D. Jones, and A. K. Patra, "Comprehensive resource use monitoring for hpc systems with tacc stats," in *2014 First International Workshop on HPC User Support Tools*. IEEE, 2014, pp. 13–21.

[13] W. Gentzsch, "Sun grid engine: Towards creating a compute power grid," in *Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE, 2001, pp. 35–36.

[14] S. G. Engine. (2011) Grid Engine man pages-Grid Engine configuration files. [Online]. Available: http://gridscheduler.sourceforge.net/htmlman/htmlman5/sge_conf.html

[15] J. Boulon, A. Konwinski, R. Qi, A. Rabkin, E. Yang, and M. Yang, "Chukwa, a large-scale monitoring system," in *Proceedings of CCA*, vol. 8, 2008, pp. 1–5.

[16] S. N. Z. Naqvi, S. Yfantidou, and E. Zimányi, "Time series databases and influxdb," *Studienarbeit, Université Libre de Bruxelles*, 2017.

[17] M. Ljubojević, A. Bajić, and D. Mijić, "Centralized monitoring of computer networks using zenoss open source platform," in *2018 17th International Symposium INFOTEH-JAHORINA (INFOTEH)*. IEEE, 2018, pp. 1–5.

[18] N. Nguyen, T. Dang, J. Hass, and Y. Chen, "Hiperjobviz: Visualizing resource allocations in high-performance computing center via multivariate health-status data," in *2019 IEEE/ACM Industry/University Joint International Workshop on Data-center Automation, Analytics, and Control (DAAC)*. IEEE, 2019, pp. 19–24.

[19] T. D. Ngan Nguyen. (2020) Time Radar. [Online]. Available: https://idatavisualizationlab.github.io/HPCC/TimeRadar/

[20] J. Hartigan, *Clustering Algorithms*. New York: John Wiley & Sons, 1975.

[21] M. Meyer, T. Munzner, and H. Pfister, "Mizbee: A multiscale synteny browser," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 897–904, Nov. 2009. [Online]. Available: http://dx.doi.org/10.1109/TVCG.2009.167

[22] D. Kammer, M. Keck, T. Gründer, A. Maasch, T. Thom, M. Kleinsteuber, and R. Groh, "Glyphboard: Visual exploration of high-dimensional data combining glyphs with dimensionality reduction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 26, no. 4, pp. 1661–1671, 2020.

[23] D. A. Keim, "Information visualization and visual data mining," *IEEE Transactions on Visualization & Computer Graphics*, no. 1, pp. 1–8, 2002.

[24] J.-l. G. Greg Roelofs and M. Adler. (2017) zlib, A Massively Spiffy Yet Delicately Unobtrusive Compression Library. [Online]. Available: https://zlib.net

[25] A. Agelastos, B. Allan, J. Brandt, P. Cassella, J. Enos, J. Fullop, A. Gentile, S. Monk, N. Naksinehaboon, J. Ogden *et al.*, "The lightweight distributed metric service: a scalable infrastructure for continuous monitoring of large scale computing systems and applications," in *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2014, pp. 154–165.

[26] D. Del Vento, D. L. Hart, T. Engel, R. Kelly, R. Valent, S. S. Ghosh, and S. Liu, "System-level monitoring of floating-point performance to improve effective system utilization," in *SC'11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2011, pp. 1–6.

[27] J. C. Browne, R. L. DeLeon, C.-D. Lu, M. D. Jones, S. M. Gallo, A. Ghadersohi, A. K. Patra, W. L. Barth, J. Hammond, T. R. Furlani *et al.*, "Enabling comprehensive data-driven system management for large computational facilities," in *SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2013, pp. 1–11.

[28] J. Li. (2020) Metrics Builder API. [Online]. Available: https://influx.ttu.edu:8080/ui/