A Scalable, Distributed Monitoring Framework for HPC Clusters Using Redfish-Nagios Integration

Ghanzanfar Ali
Texas Tech University
Jon Hass
DELL (United States)
Alan Sill
Texas Tech University
Elham Hojati
Texas Tech University
Tommy Dang
Texas Tech University
Yong Chen (yong.chen@ttu.edu)
Texas Tech University

Research Article

Keywords: DMTF Redfish Telemetry, Nagios, In-Band Monitoring, Out-of-Band Monitoring, Automation, High-Performance Computing, Data Center, Agent-less Monitoring

Posted Date: April 12th, 2023

DOI: https://doi.org/10.21203/rs.3.rs-2749012/v1

Additional Declarations: No competing interests reported.

A Scalable, Distributed Monitoring Framework for HPC Clusters Using Redfish-Nagios Integration

Ghazanfar Ali[1], Jon Hass[2], Alan Sill[1], Elham Hojati[1], Tommy Dang[1], Yong Chen[1]

[1]Department of Computer Science, Texas Tech University, 2500 Broadway, Lubbock, 79409, Texas, USA.
[2]CTO Office, Dell Technologies, 600 Congress Ave, Austin, 78701, Texas, USA.

Contributing authors: ghazanfar.ali@ttu.edu; jon.hass@dell.com; alan.sill@ttu.edu; elham.hojati@ttu.edu; tommy.dang@ttu.edu; yong.chen@ttu.edu;

Abstract

Current monitoring tools for high-performance computing (HPC) systems are often inefficient in terms of scalability and interfacing with modern data center management APIs. This inefficiency leads to a lack of effective management of the infrastructure of modern data centers. Nagios is one of the widely used industry-standard tools for data center infrastructure monitoring, which mainly includes monitoring of nodes and associated hardware and software components. However, current Nagios monitoring has special requirements that introduce several limitations. First, significant human effort is needed for the configuration of monitored nodes in the Nagios server. Second, the Nagios Remote Plugin Executor and the Nagios Service Check Acceptor are required on the Nagios server and each monitored node for active and passive monitoring, respectively. Third, Nagios monitoring also requires monitoring-specific agents on each monitored node. These shortcomings are inherently due to Nagios' in-band implementation nature. To overcome these limitations, we introduced Redfish-Nagios, a scalable out-of-band monitoring tool for modern HPC systems. It integrates the Nagios server with the out-of-band Distributed Management Task Force's Redfish telemetry model, which is implemented in the baseboard management controller of the nodes. This integration eliminates the requirements of any agent, plugin, hardware component, or configuration on the monitored nodes. It is potentially a paradigm shift in Nagios-based monitoring for two reasons. First, it simplifies communication between the Nagios server and monitored nodes. Second,

it saves computational costs by removing the requirements of running complex Nagios-native protocols and agents on the monitored nodes. The Redfish-Nagios integration methodology enables the monitoring of next-generation HPC systems using the scalable and modern Redfish telemetry model and interface.

Keywords: DMTF Redfish Telemetry, Nagios, In-Band Monitoring, Out-of-Band Monitoring, Automation, High-Performance Computing, Data Center, Agent-less Monitoring

1 Introduction

As data center technologies expand, the need to improve data center monitoring and managing technology becomes more critical. Modern data centers need more scalable monitoring tools to be able to control and manage high-performance computing (HPC) environments efficiently. Nagios [1] [2] [3] [4] is one of many widely used monitoring tools for HPC systems, and it is included as the default HPC monitoring service in some HPC stacks, such as OpenHPC stack [5]. However, there are several limitations and caveats in the current Nagios-based monitoring paradigm. First, Nagios involves significant human intervention for the definition and maintenance of remote node configurations in the Nagios Core. Second, it requires Nagios Remote Plugin Executor (NRPE) and Nagios Service Check Acceptor (NSCA) on the Nagios server and on each monitored remote node. Third, it needs monitoring-specific agents and plugins on each monitored remote node. In order to overcome these in-band limitations of Nagios, we propose and implement the integration of the Nagios Core with the state-of-the-art Out-Of-Band (OOB) Redfish telemetry model and interface [6] [7] [8] [9]. Redfish is an open and scalable industry standard, which is designed to enable data center operators to manage, monitor, and control data center resources. The key motivations and contributions related to the integration of Redfish with Nagios are described below.

1.1 Motivations and Contributions

This section describes key motivations and contributions of the prior study [10] and current study. To our knowledge, [10] was the first study that investigated and integrated Redfish with the Nagios Core. Redfish integration with Nagios is potentially a paradigm shift for Nagios-based monitoring. The key contributions of the study [10] include eliminating Nagios in-band components, enabling agent-less monitoring, and automating configuration.

1.1.1 Eliminating Nagios In-band Protocols

To monitor a node, Nagios requires Nagios-specific protocols i.e., NRPE, NSCA on the Nagios server and each monitored node. The implemented integration eliminates NRPE and NSCA by providing the monitoring functions through the BMC via the standardized Redfish application programming interface (API). This integrated monitoring service requires only the Nagios Core.

1.1.2 Enabling Agent-less Monitoring

Due to the in-band nature of the Nagios framework, Nagios requires a monitoring-specific agent (e.g., thermal monitoring) on each node. As most of the monitoring-related functions are already implemented in the baseboard management controller (BMC) and are accessible using an OOB protocol (e.g., Redfish), the integration of an OOB protocol with Nagios provides numerous benefits. First, it saves a node's computational resources considerably by offloading monitoring processing from the on-node agent to the BMC. Second, it simplifies and quickens Nagios-based monitoring significantly due to no requirement for the development, installation, and maintenance of an agent on remotely monitored nodes. Third, it minimizes the node failure risks, which can potentially happen due to in-band agent software.

1.1.3 Automating Configuration

Nagios requires the configuration of remotely monitored nodes and monitoring services. It needs tremendous human effort to perform Nagios-related configuration. Therefore, automating the configuration process is an important capability for the monitoring of large-scale modern data centers. Our method automates the generation of monitoring-related configuration information for the monitored nodes with minimum human effort.

1.1.4 Distributed Monitoring Using Redfish-Nagios Integration

Although the Redfish-Nagios integration approach has been previously described [10], a methodology for enabling distributed monitoring of large-scale HPC clusters remains undeveloped. In our current study, we address this challenge by developing a distributed monitoring framework consisting of Redfish-Nagios Servers and a central Nagios Core Server. The central Nagios Core Server distributes monitoring loads to Redfish-Nagios Servers. We evaluated the latency of a Redfish request across the Texas Tech University (TTU) Quanah HPC cluster (i.e., 467 nodes)[11]. In addition, we evaluated the impact of increasing the scale of nodes on the latency of a Redfish request. We observed that increasing the scale of monitored nodes on a Redfish-Nagios Server increases the overall latency in processing a Redfish request for a node. Based on these observations, we designed a method that allows for the estimation of the required number of Redfish-Nagios Servers which can monitor HPC nodes at scale.

1.1.5 Acquisition of Job Metrics

In addition to Redfish metrics, we added support for the acquisition of the user job metrics to map hardware telemetry with users' jobs. The objective of this mapping is to understand the behaviors of jobs and resource consumption. These behaviors are useful to optimize infrastructure per application basis.

1.1.6 Development of Web-based Monitoring Visualization Tools

While Nagios visualizes basic health telemetry, there is a lack of historical data analysis and advanced visualization tools to make use of visual analytics and provide

more informed and actionable intelligence about different perspectives and behaviors of systems and user jobs. We developed state-of-the-art web-based and machine-learning-powered monitoring visualization tools that analyze Redfish and job metrics to highlight events of significance and track the operating status of HPC nodes.

1.1.7 Saving Significant Energy

Study [12] observed that offloading low CPU utilizing (i.e., monitoring) tasks from the node's operating system to the BMC reduced the power consumption by a factor of 2.6. There is an increasing trend to perform system monitoring using OOB mechanisms [13, 14].

1.2 Organization

The rest of the study is organized as follows. Section 2 provides an overview of in-band, out-of-band, and Redfish standard. Section 3 explains the integration design, and Section 4 describes implementation aspects. Section 5 provides an experimental evaluation of the implemented method in a real HPC cluster. Section 6 explains the previous studies and we summarize this study in Section 7.

2 Background

2.1 Overview

Fig. 1 shows the overview of a typical HPC data center monitoring framework. The framework includes data center infrastructure, monitoring services, and analytics. The data center infrastructure consists of monitored hardware and software resources. The hardware resources typically constitute compute nodes, storage nodes, networking devices, and cooling and power systems. The software resources include system services and HPC applications. An example of a system service is a Slurm [15] or Univa grid engine (UGE) [16] workload manager. The HPC monitoring service acquires myriads of HPC metrics in real-time using supported HPC data center management API. The HPC metrics include thermal, power, performance, health, and job status. The health metrics provide instant status of the HPC resources in terms of normal, warning, or critical conditions. For instance, the monitoring service can identify the health status of the node and its constituent components (i.e., CPU, memory, network port, fan). The metrics related to resource consumption include CPU temperature, ambient temperature, fan speed, power consumption, CPU load, and memory usage. These utilization-related metrics are translated to normal, warning, or critical based on predefined thresholds. Furthermore, these metrics are stored in a time series database for analytic and prediction purposes. The stored metrics are exposed to analytic applications via API. The analytic applications including intelligent visual analytics and explainable machine learning are not covered in this study. The HPC metrics can be acquired via in-band and/or out-of-band protocols. These protocols are discussed below.

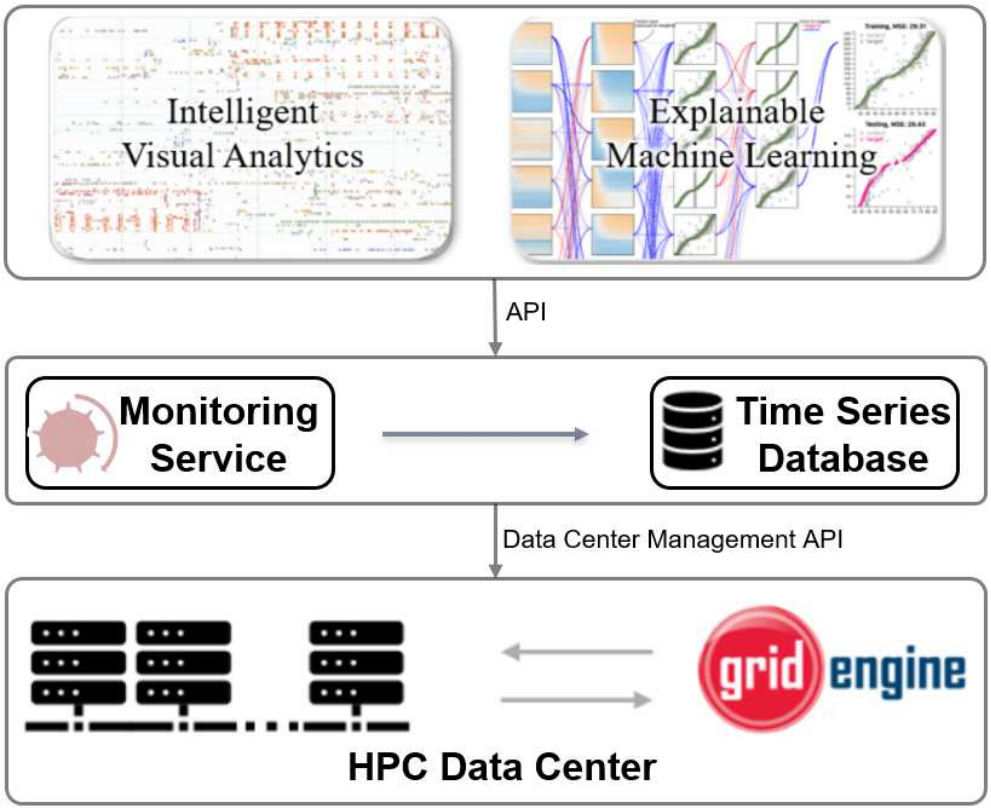


Fig. 1 Overview of HPC data center monitoring framework

2.2 In-band Monitoring and Related Overheads

In-band monitoring requires an operating system to access the target service and perform monitoring functions. The Nagios framework involves components, including NRPE and NSCA, and a plethora of monitoring check agents to perform monitoring. These components essentially communicate with the remotely monitored nodes via the operating system to acquire telemetry data. This mechanism not only complicates the monitoring of remote nodes but also causes the consumption of precious regular computational resources of the node for the processing of monitoring functions. Moreover, Nagios components and agents can potentially risk malfunctioning of the OS. Fig. 2 shows the overhead in terms of regular CPU and memory resources consumption involved in the execution of an in-band monitoring agent. This agent was developed in Golang [17] and leveraged the LIKWID HPC performance tool [18]. LIKWID used the Intel running average power limit (RAPL) interface to access power and energy consumption metrics. The agent acquires power and energy consumption at a frequency interval of one second and exposes these metrics to a monitoring service via an in-band RESTful API. The in-band monitoring agent consumes CPU up

to 2%, and memory usage is approximately 17.5 MB. The consumption of these computational resources in processing monitoring functions can interfere with and slow down the performance of the regular workloads running on the node. Therefore, it is desired to investigate and leverage out-of-band (OOB) monitoring mechanisms to save useful HPC computational resources.

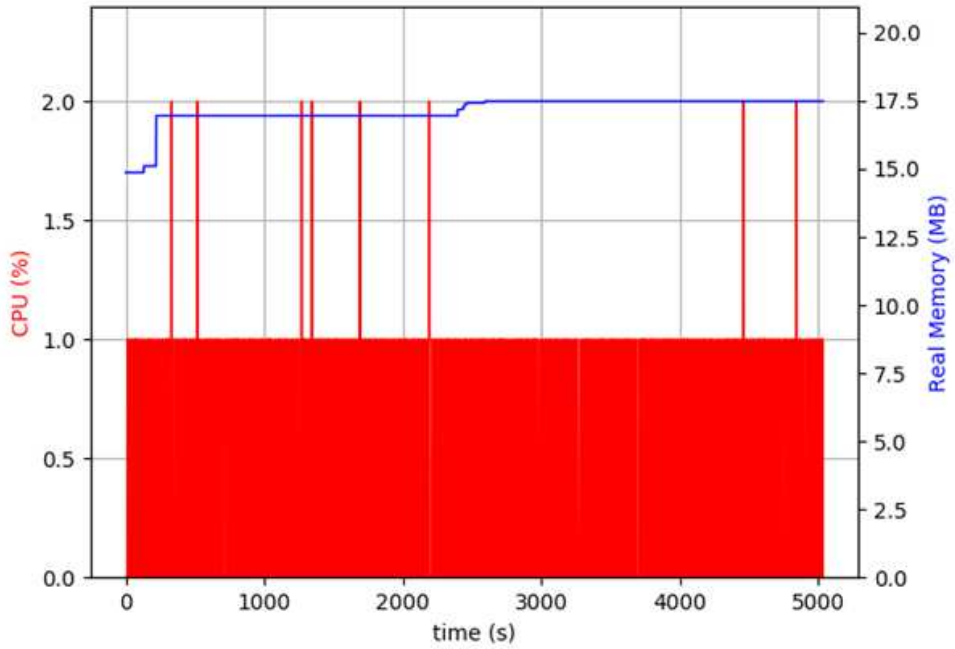


Fig. 2 Computational resources overhead of in-band monitoring

2.3 Out-of-band Monitoring and Benefits

OOB monitoring refers to a mechanism where monitoring data is acquired from a remote node via a baseboard management controller (BMC). A BMC is a specialized controller embedded in a node and often comes in the form of a system-on-chip (SoC), with its own CPU, memory, storage, and network interface card (NIC) to perform different monitoring and management functions. A BMC connects to various sensors and counters on the node to read monitoring data. It also provides other system management functions, including remote power control and interaction with the basic input/output system (BIOS). Significant progress has been made in the arena of BMC hardware and software, including Redfish and the Redfish telemetry model [6–9]. The following subsections describe OOB protocols relevant to this study.

2.3.1 Intelligent Platform Management Interface (IPMI)

IPMI is one of the most prominent initial OOB interfaces and has been widely adopted in HPC systems. It is extensively used for remote monitoring and management of the HPC, cloud [19, 20], and telecommunication infrastructure. IPMI is broadly used to perform remote node power control and acquires telemetry data, such as node power consumption and thermal condition[21]. While IPMI has been widely adopted in the initial data center for remote system management and monitoring, it has notable disadvantages that include security issues[22], scalability, and complexity due to being a bit-wise protocol.

2.3.2 Redfish

Redfish [23] [6] [7] [8] [9] is a state-of-the-art OOB standard, which is implemented in the BMC of a node. It is designed to deliver simple and secure management for data center infrastructure. Redfish leverages common Internet and web service standards to expose information directly to the modern data center management and monitoring toolchain. Redfish consists of an interface protocol and a data model. The data model is expressed in terms of a standardized, machine-readable schema, with the payload of the messages being expressed in JavaScript Object Notation (JSON). It is a hypermedia API, which implies that subordinate resources are discoverable from the uniform resource identifiers (URI) of top resources. The root URI for the Redfish API is http://address/redfish/v1/, where the address can be a name or IP address of the Redfish-enabled endpoint, and redfish/v1/ refers to the Redfish resource. Every resource corresponds to a specific Redfish URI. Since Redfish is based on RESTful principles, it supports `Create`, `Read`, `Update`, and `Delete` operations. To perform these operations, Redfish supports `POST` to create resources, `GET` to read data, `PATCH` to change one or more properties on a resource, and `DELETE` to remove a resource permanently. Fig. 3 [23] provides an overview of Redfish. The major objects involved in Redfish include systems, managers, and chassis. Systems represent the logical view of the server and consist of CPU and memory. Managers are essentially a BMC, an enclosure manager, or another component that manages the infrastructure. A chassis provides the physical view and includes racks, enclosures, and the blades within them. The chassis also can include other chassis and consists of a variety of sensors and fans. The grouping of similar resources is defined as a collection. For example, a group of systems, a group of BMCs, and a group of chassis can be represented as `SystemCollection`, `ManagersCollection`, and `ChasissCollection`, respectively. A collection returns the number of resources in the field Members@odata.count and URIs of those resources are listed in the Members field. We acquired the node metrics, including power usage, CPU temperature, fan speeds, power status, OS status, and health of BMC, node, CPU, memory, and fan. Redfish as a successor of IPMI supports rich data center management and control capabilities. Table 1 summarizes key differences between Redfish and IPMI.

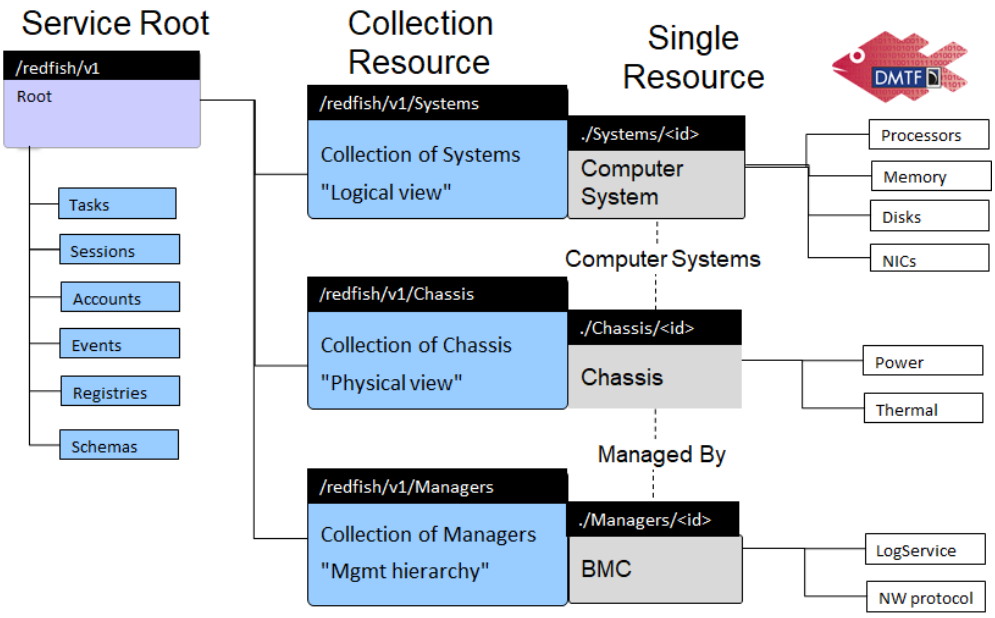


Fig. 3 DMTF Redfish API Basic View [23]

Table 1 Comparison Between Redfish and IPMI

Redfish	**IPMI**
Hypertext transfer protocol (HTTP) over transmission control protocol (TCP)	Remote control and management protocol (RCMP) over user datagram protocol (UDP)
RESTful Architecture	No RESTful support
Scalable due to RESTful nature	Limited scaleability
Supports for IT and non-IT equipment	Support limited to IT equipment
Human-readable format (i.e., JSON)	Bit-wise protocol
Reliable (due to TCP)	Unreliable (due to UDP)
Secure (HTTPS, session)	No credible security mechanisms
Supports telemetry model	No support for telemetry model

3 Integration Methodology

This section describes the proposed Redfish-Nagios tool, a Nagios monitoring tool based on DMTF Redfish telemetry model and standard. As explained in the background section, in-band-based monitoring (Fig.4) introduces several limitations. For example, it needs (1) a significant portion of system resources such as CPU and memory to process the in-band-based monitoring functions, (2) enormous human effort in performing manual configuration, and (3) implementation, deployment, and maintenance of protocols and agents used in in-band based monitoring. Currently, Nagios suffers from these limitations.

In this paper, we present the Redfish-Nagios integrated tool. This tool leverages the capabilities of OOB communication (Fig.5) to overcome the limitations of the current Nagios framework and improves the performance and the scalability of the monitoring tool for modern, scalable data centers. Different communication methods (i.e., in-band and out-of-band), Nagios-Redfish integrated tool architecture, and interworking between Nagios and Redfish are described below.

3.1 In-band Communication

In-band communication refers to a communication paradigm that requires a regular operating system to access the target service. The in-band communication mechanism is shown in Fig.4. As depicted in Fig.4, the protocol-specific in-band endpoint

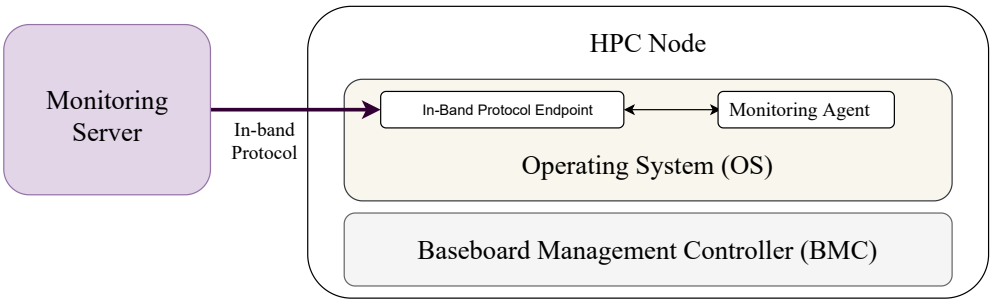


Fig. 4 In-band communication mechanism

and monitored entity-specific monitoring agent is implemented and deployed on each monitored node. This process involves enormous human efforts in terms of implementation, deployment, and maintenance. In this arrangement, the monitoring server initiates the acquisition of monitoring data from the monitored node. The in-band protocol endpoint implements the protocol functions (e.g., SNMP). The in-band agent is responsible for implementing a particular monitoring function such as temperature data acquisition from a thermal sensor. Nagios framework is a typical example of in-band monitoring. It collects metrics from the remote monitored node using NRPE, which is installed on each monitored node.

3.2 OOB Communications

OOB communication does not require an OS for its interaction with the target service. The OOB communication mechanism is shown in Fig.5. As depicted in Fig.5, the monitoring server bypasses the node OS and directly communicates with the node's BMC. The BMC enables communication via OOB protocols, such as IPMI and Redfish. As explained in Section 1, the overall goals of this study are to enable agent-less monitoring, automating configuration, eliminating Nagios in-band components, and leveraging cloud tools and technologies to monitor and manage modern data centers.

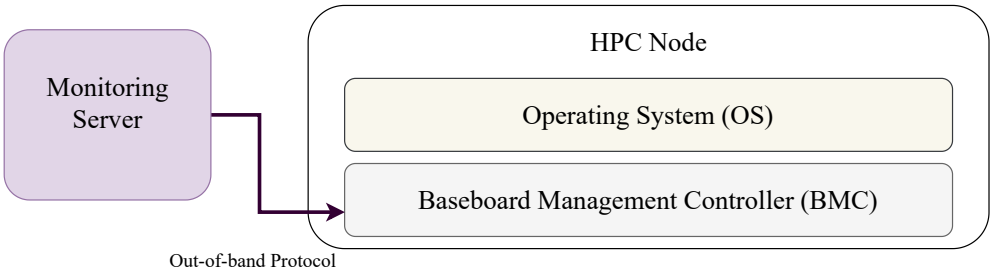


Fig. 5 OOB communication mechanism

Redfish API, a state-of-the-art OOB-based model, and the specification are used to achieve these objectives.

3.3 Redfish-Nagios Tool Architecture

Fig. 6 depicts the proposed Redfish-Nagios tool architecture. The Redfish-Nagios tool consists of three layers: 1) a Nagios Core, a configurable monitoring framework, which performs check scheduling, check execution, check processing, event handling, and alert management; 2) an abstraction layer between the Nagios Core and the HPC infrastructure, which includes plugins, executables, or scripts; and 3) the HPC infrastructure, consisting of the Redfish-enabled nodes, node hardware components (e.g., BMC, CPU, memory, storage), and services. The Redfish-enabled node implies that the BMC of the node supports Redfish. The Nagios Core communicates with Redfish plugins via an internal interface. In turn, the plugins communicate with the monitored nodes via Redfish. This integrated architecture does not require any Nagios protocols, such as NRPE, NSCA, or any additional agent running on the Nagios server and monitored nodes. The integration of the Nagios Core with the open industry standard Redfish aims to enable standardized, efficient, generic, and automated Nagios-based data center monitoring. The Redfish-Nagios tool supports seven Redfish-based plugins for Nagios as listed in Table 2. The above plugins use the following Redfish URIs

Table 2 Redfish Plugins for Nagios

Plugin Name	**Description**
check_BMC	Acquires BMC health
check_host	Acquires node health
check_CPU	Acquires CPU health
check_memory	Acquires memory health
check_fans	Acquires fan health and speed
check_temperature	Acquires CPU temperature
check_power_usage	Acquires node power consumption

to acquire different metrics, as shown in Table 3. Overall, Table 4 shows 15 and three metrics acquired via Redfish API and UGE, respectively.

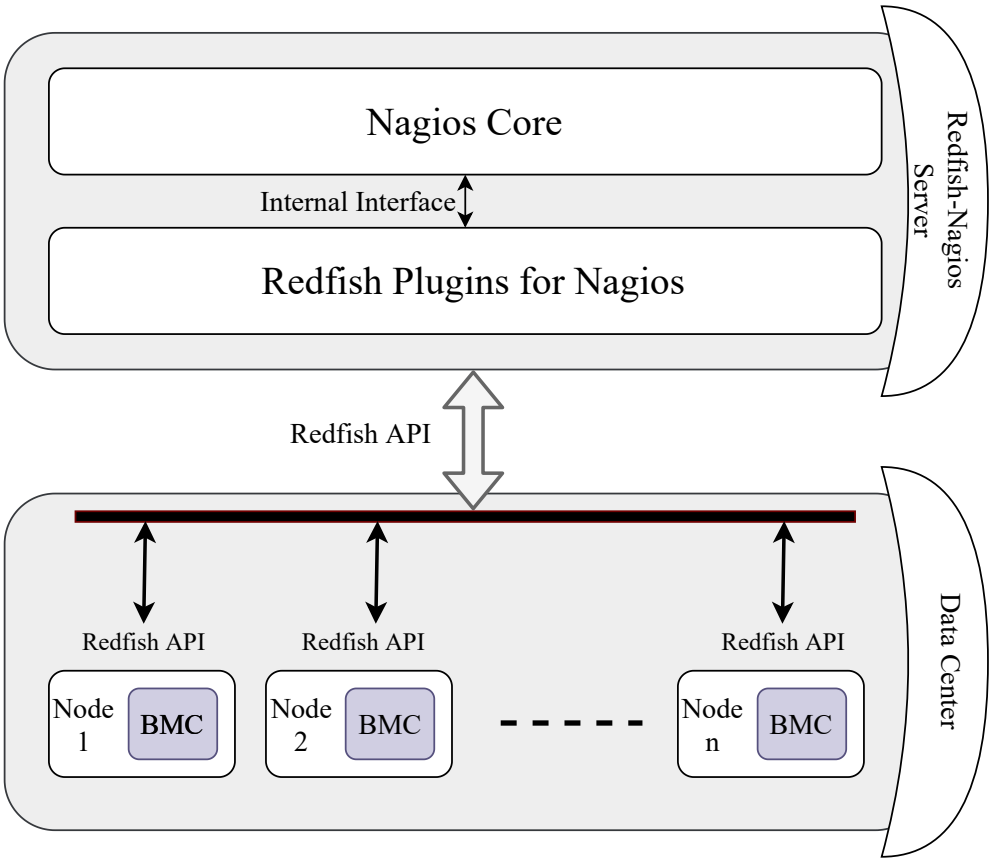


Fig. 6 Redfish-Nagios Tool Architecture

Table 3 Redfish URIs

Redfish URI	**Metrics**
https://{bmc_ip}/redfish/v1/Systems	Node health status
https://{bmc_ip}/redfish/v1/Chassis/Thermal	Node thermal
https://{bmc_ip}/redfish/v1/Systems/Power	Node power consumption
https://{bmc_ip}/redfish/v1/Managers	BMC health status

3.4 Inter-working Between the Nagios Core and Redfish States

The Redfish-Nagios tool performs monitoring functions using Redfish-based plugins. The plugins can return monitoring data as health status or numeric data. When the monitoring data denotes a health status of a resource, the state is determined according to the health status property received in Redfish. Note that there is a one-to-one mapping between Redfish and Nagios health status properties. Therefore, there is no need to apply threshold-based calculations to determine resource health status and the resource health statuses are directly translated as shown in Table 5.

Table 4 Redfish and UGE metrics used in Redfish-Nagios tool

Metric Name	Description and Source
host_health	node operational status (Redfish)
CPU_health	CPU operational status (Redfish)
memory_health	memory operational status (Redfish)
bmc_health	bmc operational status (Redfish)
fan1_health	fan1 operational status (Redfish)
fan2_health	fan2 operational status (Redfish)
fan3_health	fan3 operational status (Redfish)
fan4_health	fan4 operational status (Redfish)
fan1_speed	fan1 speed (revolutions per minute (RPM)) (Redfish)
fan2_speed	fan2 speed (RPM) (Redfish)
fan3_speed	fan3 speed (RPM) (Redfish)
fan4_speed	fan4 speed (RPM) (Redfish)
cpu1_temperature	CPU1 (socket 1) temperature (degrees Celsius) (Redfish)
cpu2_temperature	CPU2 (socket 2) temperature (degrees Celsius) (Redfish)
power_usage	Node power consumption (Watts)(Redfish)
cpu_usage	CPU load (%) (UGE)
memory_usage	memory usage (%) (UGE)
job_data	job ID, user ID, hosts ID (UGE)

Table 5 Redfish and Nagios Monitoring Status

Redfish Status	Nagios Status	Description
Ok	OK	Working correctly
Warning	WARNING	Working, but needs attention
Critical	CRITICAL	Not working correctly or requires attention
Unknown	UNKNOWN	Plugin was unable to determine the status

When the monitoring data is a numeric value, the value is translated to one of three possible Nagios states as shown in Table 5 based on a predefined threshold. The conversion of numeric monitoring data to a status based on a threshold is an extremely useful practice to change monitoring data into a useful insight related to the operational status of an HPC resource. In this way, the administrator is provided with visual analytics via a Nagios dashboard that can provide a better status overview of the cluster rather than reading and trying to understand all metrics quantitatively. For example, an HPC administrator might unintentionally miss a high CPU temperature, even when the CPU temperature reaches or goes beyond a critical point. Having a threshold-based categorization of the resource health status is more useful because the tool will be able to detect and show whether a resource is changing from a normal condition to a warning or a critical status.

The code listing 1 shows the Bash script for the Redfish check_temperature plugin. The detailed source code for the Redfish plugins is available on GitHub [24].

3.5 Distributed Monitoring Using Redfish-Nagios

HPC data center monitoring can be challenging when monitoring a large number of nodes. To monitor a large-scale HPC cluster, we designed a distributed monitoring

```
#!/bin/bash

chassis_uri=$(curl -k https://$1/redfish/v1/Chassis/ -u root:redfish | jq '.
    Members|.[0]|." @odata.id"' | sed 's/"//g')

health=$(curl -k https://$1$chassis_uri/Thermal/ -u root:redfish | jq '.
    Temperatures[]|(.Name ,.ReadingCelsius ,(.Status|.Health))' | sed 's/"//g'
    )

if [[ $health == *"Critical"* ]]; then
  severity='Critical'
elif [[ $health == *"Warning"* ]]; then
  severity='Warning'
elif [[ $health = *"OK"* ]]; then
    severity='OK'
fi
if [ "$severity" == 'OK' ]; then
                echo $health
                exit 0
elif [ "$severity" == 'Warning' ]; then
                echo $health
                exit 1

elif [ "$severity" == 'Critical' ]; then
                echo $health
                exit 2
else
                echo "UNKNOWN-Plugin was unable to determine the status for
    the host CPU temperatures!"
                exit 3
fi
```

Listing 1 Code Check_Temperature Plugin

framework using the Redfish-Nagios tool as shown in Fig. 7. The distributed monitoring framework consists of three layers: (1) the central Nagios Core Server layer; (2) the Redfish-Nagios Server layer; and (3) the target-monitored HPC nodes. The central Nagios Core Server performs two key functions to enable distributed monitoring.

The first key function of the central Nagios Core Server is to distribute the Nagios commands to Redfish-Nagios Servers using `CmdDist` [25]. However, the estimation of the monitoring load and required Redfish-Nagios Servers is another challenge. We address this issue by taking into consideration three aspects: (a) the granularity of monitoring intervals, (b) the number of Redfish requests, and (c) the number of monitored nodes. A strict monitoring interval is crucial for the collection of time-sensitive metrics. However, this interval can not be ensured when a single Redfish-Nagios Server is used to handle the monitoring load at a large scale. Therefore, a methodology is required to scale monitoring based on the load. This methodology consists of three major steps. As a first step, we measure the end-to-end average latency in handling a single Redfish request (with a round-trip time (RTT)) across all nodes (N) in a cluster using Eq. 1.

$$Latency_{Avg} = \frac{1}{N}\sum_{i=1}^{N} RTT_i + ComputeTime_i \tag{1}$$

The average latency is influenced by the number of monitored nodes, the time spent processing the requests, and the capability of computing and communication resources of the monitoring server. The next step involves the calculation of load intensity for a

Redfish request using RTT (for a single node) and latency (across all nodes) as shown in Eq. 2.

$$Load_{Intensity} = \frac{Latency_{Avg}}{RTT} \tag{2}$$

A load intensity with a value close to one will be ideal and indicates that the monitoring server is not overloaded. This intensity allows a monitoring interval equal to or greater than the RTT of the Redfish request. However, a higher load intensity means the monitoring server is overloaded and it will also affect the monitoring interval accordingly. Finally, using the load intensity ($Load_{Intensity}$) (Eq. 2), we can estimate the number of Redfish-Nagios Servers ($Servers$) required to monitor Redfish requests (R) with an average RTT for a given monitoring interval ($Interval$) as shown in Eq. 3.

$$Servers = \frac{R \times RTT \times Load_{Intensity}}{Interval} \tag{3}$$

If the value of the $Servers$ is less than or equal to one, a single Redfish-Nagios Server is sufficient to handle the monitoring load. On the other hand, if the value of the $Servers$ is greater than one, the above equation will determine the required number of Redfish-Nagios Servers.

The second key function of the central Nagios Core Server is to provide a global view of the entire HPC center by obtaining statuses from each Redfish-Nagios Server. The target-monitored HPC nodes are configured to be monitored by a single Redfish-Nagios Server only. In other words, the nodes cannot be monitored by multiple Servers simultaneously. The interaction between Redfish-Nagios Servers and HPC nodes is described in Section 3.3. Each Redfish-Nagios Server handles notifications, statuses, and reports locally. Furthermore, each Redfish-Nagios Server is configured to transfer check results (status information) back to the central Nagios Core Server using Nagios distributed monitoring internal interfaces, which are publicly available at the Nagios Exchange. For example, `check_nagios_summary` [26], `check_remotenagios` [27], and `check_summary` [28] can be used by the Redfish-Nagios Servers to report statuses to the Nagios Core Server. These interfaces allow the central Nagios Core Server to have a collective view of the cluster monitored by each Redfish-Nagios Server.

4 Implementation

This section provides the implementation details of the Redfish-Nagios tool. First, it explains the testbed used to evaluate the Redfish-Nagios tool. Then, it describes the hardware and software configurations of the Redfish-Nagios server. After that, it demonstrates the internal working of the Redfish-Nagios server.

4.1 Testbed for Redfish-Nagios Tool

We used the Quanah cluster [11] as a testbed for the evaluation of the Redfish-Nagios tool. The cluster consists of 467 nodes, and each node is based on the Intel XEON processor architecture and consists of 36 cores. The BMC uses the integrated Dell Remote Access Controller 8 (iDRAC8) [29], which implements the Redfish API [23]

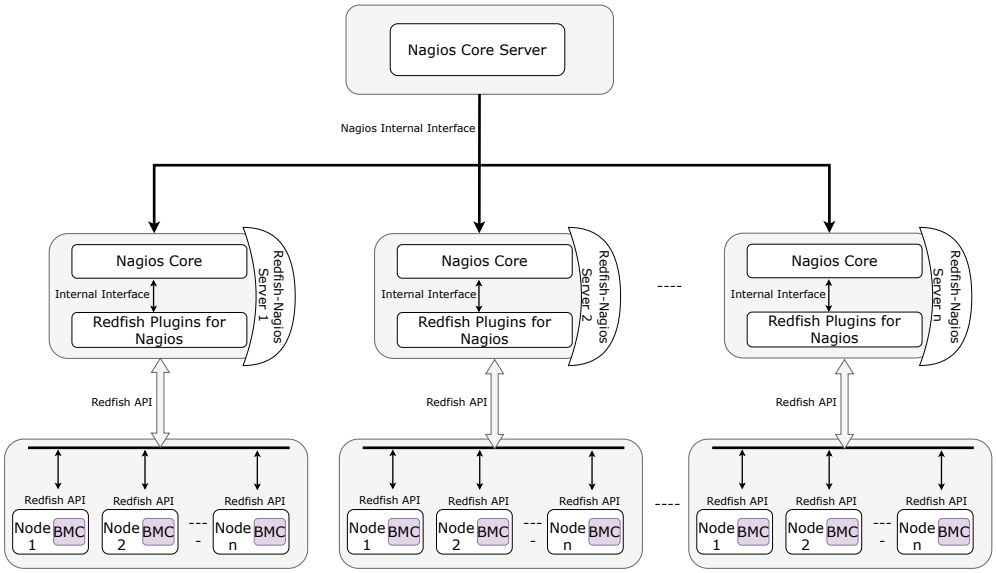


Fig. 7 Distributed Monitoring using Redfish-Nagios

Table 6 Host Hardware Specifications

CPU	2 x 4 cores Intel Xeon(R) E5540 @ 2.53GHz
RAM	23 GB DDR3
STORAGE	2TB HDD
NETWORK	1Gbit/s, Broadcom NetXtreme II

to deliver remote management and monitoring capabilities. The operating system of the compute nodes in the cluster is Linux CentOS 7.6.

4.2 Redfish-Nagios Server Configuration

The Redfish-Nagios integrated monitoring service requires Nagios Core 4.4.0, Nagios configurations, and Redfish plugins for the Nagios Core. This setup does not require NRPE, NSCA, or any agent as required in the current Nagios framework setup. Table 6 provides hardware specifications of the host running Nagios monitoring service integrated with Redfish API.

4.3 Redfish-Nagios Tool Deployment

The integrated monitoring service is deployed on the CentOS 7.6 Linux server, and Redfish plugins are implemented using Bash. Nagios Core operational behavior is configurable by editing the configuration parameters in `nagios.cfg`. To monitor a node and the components attached to the node, Nagios requires configuration information (e.g., IP address) of the BMC of the monitored node. The acquisition of the IP addresses of BMCs and the related configuration information into the Nagios Core

were performed automatically. This setup consists of 467 nodes with Redfish-enabled BMCs and nine services or monitoring checks per node. These services are defined as commands in the Nagios Core (i.e., `commands.cfg`), which can be directly invoked by the Nagios Core.

Historically, the Nagios Core requires configuration information for each monitored node and service. These configurations are performed manually, making Nagios-based monitoring deployment difficult and error prone, especially on a large scale. The implemented Redfish-Nagios tool automates these configurations. This process consists of two steps. First, the lists of node names and IP addresses of BMCs are acquired. We obtained these IP addresses and node names from the management node and stored them in the `nagios_node_config.conf` file. The nodes, which need to be configured for Nagios monitoring, are listed in the [$NodesInfo$] section. Each entry corresponds to a node name and IP address of the BMC, which are separated by a colon as shown in listing 2.

```
[NodesInfo]
new_ip_list = compute-1:10.9.1.1,compute-2:10.9.1.2,compute-3:10.9.1.3
```

Listing 2 HPC node information

After acquiring the list of node names and BMC IP addresses, the script `nagnodeconfig.py` reads that list from the `nagios_node_config.conf` file and defines node information and related services in `hosts.cfg` file. The example configuration outcome of a node and service are shown in listing 3 and listing 4, respectively.

```
define host{
    use     linux-server
    host_name     Compute1
    alias     localhost
    address     10.9.1.1
    }
```

Listing 3 Node configuration information

```
define service{
    use     local-service
    host_name     Compute1
    service_description     check temperature
    check_command     check-temperature
    }
```

Listing 4 Node's service configuration information

4.4 Internal Working of the Redfish-Nagios Tool

In order to make monitoring efficient and fully utilize the system computing capability of a multi-core HPC node [30], the monitoring workload is distributed evenly among the available cores [31]. To monitor a metric using Redfish API across 467 nodes in the Quanah cluster, the Redfish-Nagios server initiated 467 Redfish requests and parallelized them among 8 CPU cores of the server. Fig. 8 shows the parallelization of Redfish requests for monitoring power usage for 467 nodes in the Quanah cluster. The seven cores handle 58 requests each and 8^{th} core handles remaining $(58+3=61)$ requests.

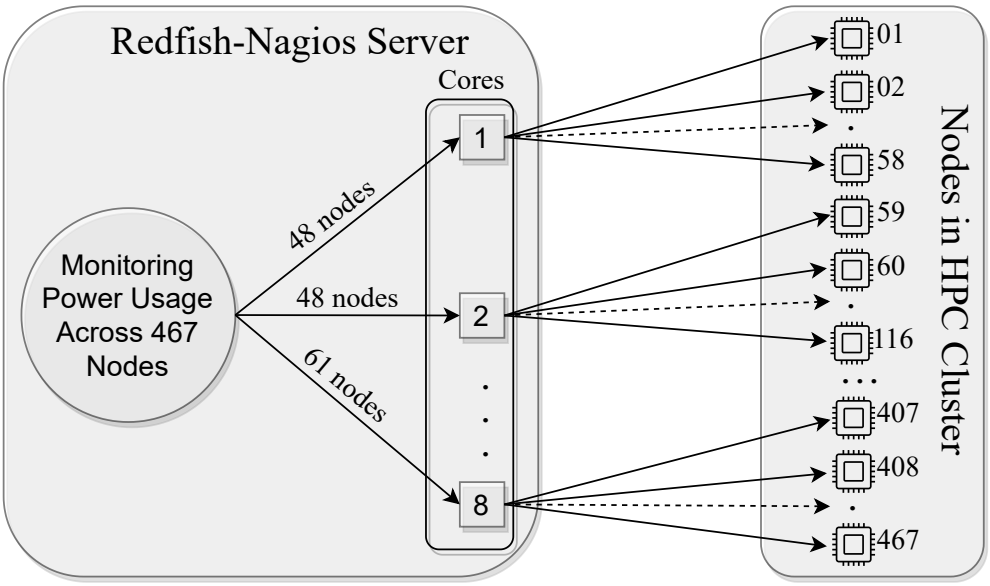


Fig. 8 Parallelization of Redfish requests in Redfish-Nagios Server, in monitoring power usage across 467 nodes in the Quanah cluster.

5 Evaluation with Real HPC Cluster

We evaluated the integration methodology of Redfish-Nagios using the Quanah cluster at the High-Performance Computing Center (HPCC) of Texas Tech University [11]. The results presented in this section include the performance evaluation of the Redfish-Nagios Server and monitoring visualization using this tool.

5.1 Redfish-Nagios Server Performance

Our Redfish-Nagios Server ran 3,269 monitoring checks (467 nodes X 7 checks) at a rate of two-minute monitoring intervals. With this monitoring workload on the Redfish-Nagios Server, the average CPU load was ∼69%, the memory usage was 3.04 GB, and the network bandwidth was 2.13 Mbps. The Redfish-Nagios tool uses Redfish to perform monitoring services; therefore, it is important to evaluate the performance of the Redfish API.

5.1.1 Redfish Request Processing Time Across HPC Cluster

Fig. 9 shows the latency of a Redfish power usage request from the Redfish-Nagios Server to 467-node Texas Tech Quanah cluster. Each dot represents an HPC node. The Redfish request is sent to each node using the asynchronous parallel method. The x-axis represents the round-trip time (RTT), in seconds, of the Redfish request to the HPC nodes. The node-to-node RTT is variable and is between 4.13 and 5.82 seconds. Hence, the maximum latency (see 1) of a Redfish request across the HPC cluster is not

more than six seconds. We observed that the RTT of the Redfish power usage request to a single node is also not more than six seconds. Thus, the load intensity (see 2) of the request is approximately one. As a result, a single Redfish-Nagios Server (see 3) is sufficient to monitor power usage for the Quanah cluster at a minimum monitoring interval of six seconds.

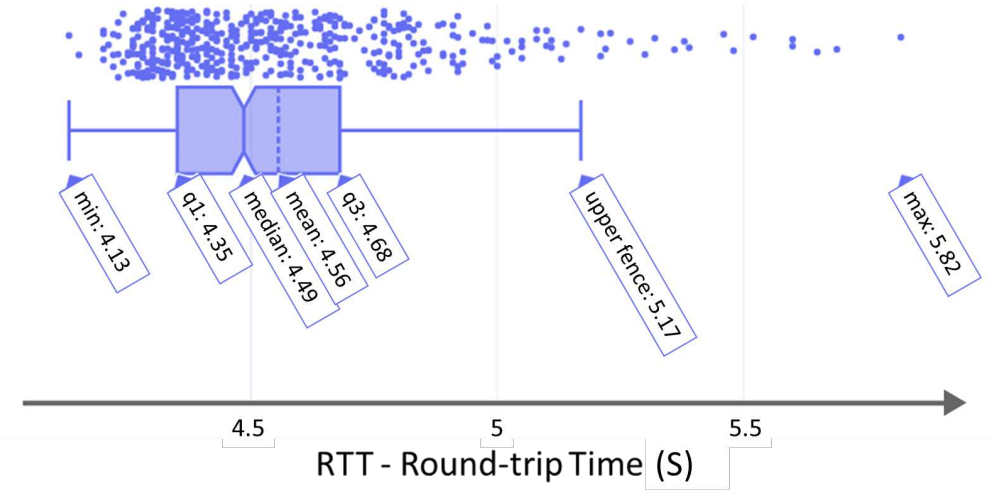


Fig. 9 The latency of a Redfish's power usage request from the Redfish-Nagios Server to the TTU's Quanah HPC cluster (i.e., 467 nodes). Each dot represents an HPC node. The Redfish request is sent to each node using the asynchronous parallel method.

5.1.2 Scalable Distributed Monitoring of HPC Cluster

We exemplified the distributed monitoring of the 467-node Quanah cluster using different scenarios. Fig. 10 shows the scenario where the RTT of a typical Redfish request is six seconds, the sampling interval is 60 seconds, and 10 Redfish-based metrics are required to be acquired across the Quanah cluster within the given sampling interval. It is assumed that a BMC handles one Redfish request at a time. "Monitoring Service" refers to the Redfish-Nagios service that initiates cluster-wide Redfish requests (parallel) at a monitoring interval of 60 seconds. "467 BMCs" refers to the 467 nodes in the Quanah cluster that handle the received Redfish requests. The goal is to estimate the number of Redfish-Nagios Servers required to handle this monitoring load. Empirical observation (see Fig. 9) showed that a latency (1) of approximately six seconds is involved in processing a single Redfish request across the 467 nodes in the cluster. Based on the Redfish request's RTT and the average latency, the load intensity (2) is calculated as approximately one. In this way, we can estimate the required number of Redfish-Nagios Servers using the above parameters in Eq. 3 (i.e., $\frac{10\times6\times1}{60} = 1$). Thus, a single Server is sufficient to handle 10 Redfish requests at a sampling interval

of 60 seconds or higher. However, when the sampling interval is reduced to the average latency, i.e., six seconds, and other parameters remain unchanged, the required Redfish-Nagios Servers are equal to 10 (i.e., $\frac{10\times6\times1}{6} = 10$). Overall, it is anticipated that the number of monitored nodes, the number of Redfish requests, the RTT of the requests, and the sampling interval directly influence the number of Redfish-Nagios Servers required in the distributed monitoring.

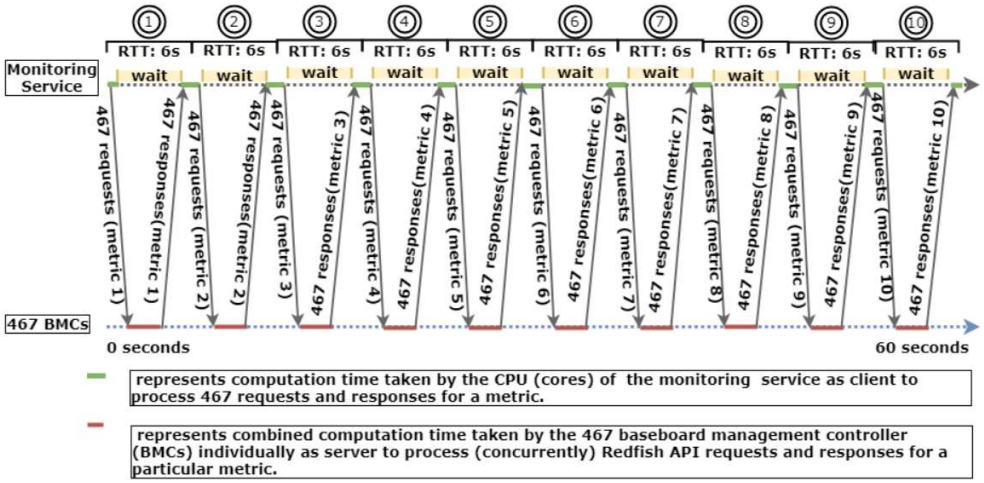


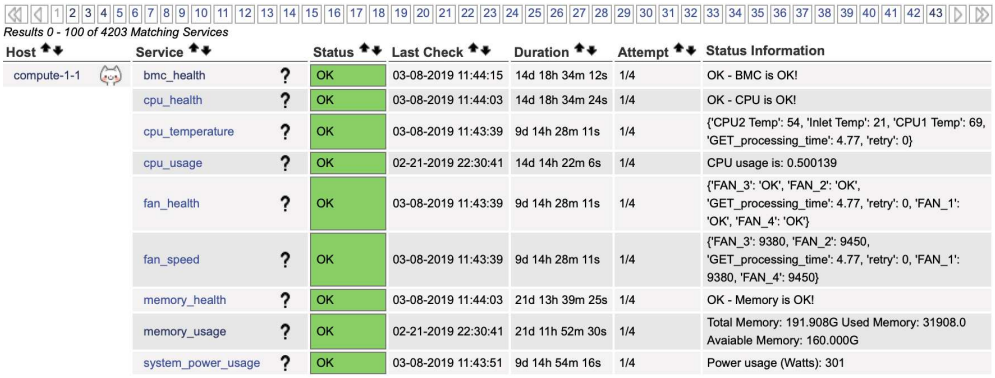
Fig. 10 Number of Redfish Requests Supported per Monitoring Interval

5.2 Monitoring Visualization Using the Redfish-Nagios Tool

The monitoring information acquired via the Redfish-Nagios tool is visualized at the component, node, and cluster levels.

5.2.1 Component Level

Fig. 11 shows the monitoring information of 9 components (e.g., hardware resources, software services) associated with an HPC node. Redfish collects metrics for seven components, including cpu_temperature, system_power_usage, fan_health, fan_speed, memory_health, cpu_health, and bmc_health. Nagios defines four statuses for the components: Ok, Warning, Critical, and Unknown. Ok, Warning, and Critical match with the Redfish health statuses, so Unknown is not taken into consideration. Some services also return quantitative data, such as CPU temperature, fan speed, and node power usage. The numeric data is translated into a status such as Ok, Warning, or Critical based on a threshold.



1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43

Results 0 - 100 of 4203 Matching Services

Host	Service	Status	Last Check	Duration	Attempt	Status Information
compute-1-1	bmc_health	OK	03-08-2019 11:44:15	14d 18h 34m 12s	1/4	OK - BMC is OK!
	cpu_health	OK	03-08-2019 11:44:03	14d 18h 34m 24s	1/4	OK - CPU is OK!
	cpu_temperature	OK	03-08-2019 11:43:39	9d 14h 28m 11s	1/4	{'CPU2 Temp': 54, 'Inlet Temp': 21, 'CPU1 Temp': 69, 'GET_processing_time': 4.77, 'retry': 0}
	cpu_usage	OK	02-21-2019 22:30:41	14d 14h 22m 6s	1/4	CPU usage is: 0.500139
	fan_health	OK	03-08-2019 11:43:39	9d 14h 28m 11s	1/4	{'FAN_3': 'OK', 'FAN_2': 'OK', 'GET_processing_time': 4.77, 'retry': 0, 'FAN_1': 'OK', 'FAN_4': 'OK'}
	fan_speed	OK	03-08-2019 11:43:39	9d 14h 28m 11s	1/4	{'FAN_3': 9380, 'FAN_2': 9450, 'GET_processing_time': 4.77, 'retry': 0, 'FAN_1': 9380, 'FAN_4': 9450}
	memory_health	OK	03-08-2019 11:44:03	21d 13h 39m 25s	1/4	OK - Memory is OK!
	memory_usage	OK	02-21-2019 22:30:41	21d 11h 52m 30s	1/4	Total Memory: 191.908G Used Memory: 31908.0 Avaiable Memory: 160.000G
	system_power_usage	OK	03-08-2019 11:43:51	9d 14h 54m 16s	1/4	Power usage (Watts): 301

Fig. 11 Node's Component Level Monitoring Visualization

5.2.2 Node Level

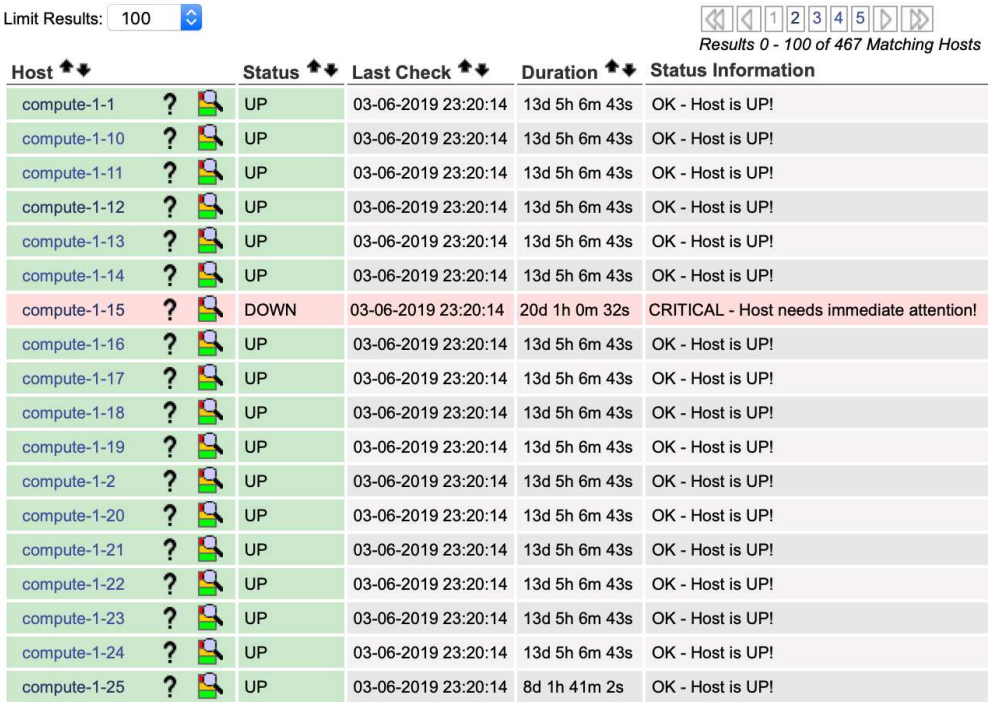
Fig. 12 shows the monitoring information of nodes including node name, last check, duration, and status information. Redfish provides node status as Ok, Warning, or Critical. On the other hand, Nagios shows node status as UP or DOWN. Redfish Ok and Warning are translated as Nagios UP, and Redfish Critical is translated as Nagios DOWN. A row in green color indicates the node is UP and running correctly, while a row in red color shows the node is DOWN due to a malfunction in one or more of its components.

5.2.3 Cluster Level

Node-level visualization is not sufficient to provide a high-level summary of the large-scale cluster. Fig. 13 provides a comprehensive view of the HPC cluster in terms of nodes' status (Ok, Warning, or Critical). Each HPC node is shown by a small circle, labeled with its configured name. The node labels are shown in black, yellow, or red. A node with a black label reflects that the node and its related components are working appropriately, while a node with a yellow label indicates that one or more components of the node are not working properly and need attention to resolve the underlying problem. A node with a red label shows that the node is in a critical condition, which means it is not functioning and needs immediate attention for problem rectification.

5.3 Web-based Monitoring Visualization Tools

While Nagios visualizes the basic health telemetry at the component, node, and cluster level (as depicted in Figures 11, 12, and 13), there is a lack of advanced visualization tools that can provide a more informed and actionable intelligence about the HPC infrastructure. However, it is almost impossible to design a single visualization tool to analyze the Redfish and job scheduler (UGE) metrics (acquired via the Redfish-Nagios tool) from different perspectives. Therefore, our goal is to provide various web-based advanced visualization tools utilizing multiple statistical and machine-learning approaches. We developed several innovative visual analytics tools,



Limit Results: 100

1 2 3 4 5

Results 0 - 100 of 467 Matching Hosts

Host	Status	Last Check	Duration	Status Information
compute-1-1	UP	03-06-2019 23:20:14	13d 5h 6m 43s	OK - Host is UP!
compute-1-10	UP	03-06-2019 23:20:14	13d 5h 6m 43s	OK - Host is UP!
compute-1-11	UP	03-06-2019 23:20:14	13d 5h 6m 43s	OK - Host is UP!
compute-1-12	UP	03-06-2019 23:20:14	13d 5h 6m 43s	OK - Host is UP!
compute-1-13	UP	03-06-2019 23:20:14	13d 5h 6m 43s	OK - Host is UP!
compute-1-14	UP	03-06-2019 23:20:14	13d 5h 6m 43s	OK - Host is UP!
compute-1-15	DOWN	03-06-2019 23:20:14	20d 1h 0m 32s	CRITICAL - Host needs immediate attention!
compute-1-16	UP	03-06-2019 23:20:14	13d 5h 6m 43s	OK - Host is UP!
compute-1-17	UP	03-06-2019 23:20:14	13d 5h 6m 43s	OK - Host is UP!
compute-1-18	UP	03-06-2019 23:20:14	13d 5h 6m 43s	OK - Host is UP!
compute-1-19	UP	03-06-2019 23:20:14	13d 5h 6m 43s	OK - Host is UP!
compute-1-2	UP	03-06-2019 23:20:14	13d 5h 6m 43s	OK - Host is UP!
compute-1-20	UP	03-06-2019 23:20:14	13d 5h 6m 43s	OK - Host is UP!
compute-1-21	UP	03-06-2019 23:20:14	13d 5h 6m 43s	OK - Host is UP!
compute-1-22	UP	03-06-2019 23:20:14	13d 5h 6m 43s	OK - Host is UP!
compute-1-23	UP	03-06-2019 23:20:14	13d 5h 6m 43s	OK - Host is UP!
compute-1-24	UP	03-06-2019 23:20:14	13d 5h 6m 43s	OK - Host is UP!
compute-1-25	UP	03-06-2019 23:20:14	8d 1h 41m 2s	OK - Host is UP!

Fig. 12 Node Level Monitoring Visualization.

including `Hiperview`, `HiperVR`, `ClusterView`, `Phase Space`, `ParallelCoordeinates`, `TimeRadar`, `Heatmap`, `Jobviewer`, `Scagnostics Viewer`, `Spiral Layout`, `JobSwarm`, `SankeyViewer`, `Connected Scatterplot`, `Job Net`, `Dynet3D`, `JobPCA`, `Job Squeue`, `JobPower`, and `PowerMap`. Each of these visualization techniques addresses a specific use case of visual analytics. The selection of visualization techniques is based on the scope and visualization use case. We describe `Hiperview`, `ParallelCoordeinates`, and `TimeRadar` in more detail below.

5.3.1 Visualization Using HiperView

Hiperview is a visual analytics tool that characterizes and visualizes the health status of HPC nodes [32]. Fig. 14 shows the main graphical interface of the HiperView visualization which consists of a radar charts view and a main view using heatmaps. The key advantage of this technique is that it allows for the visualization of time series data by mimicking the physical layout of the HPC data center. Radar charts show a summary of different metrics at a timestamp. Each host shows two heatmaps that represent the temperature status for CPU1 and CPU2. The right-most cell shows the most recent temperature. For example, the temperature of CPU 2 of Host 13 in Rack 3 is in critical status for the last 4 time steps. This visualization provides ride-through

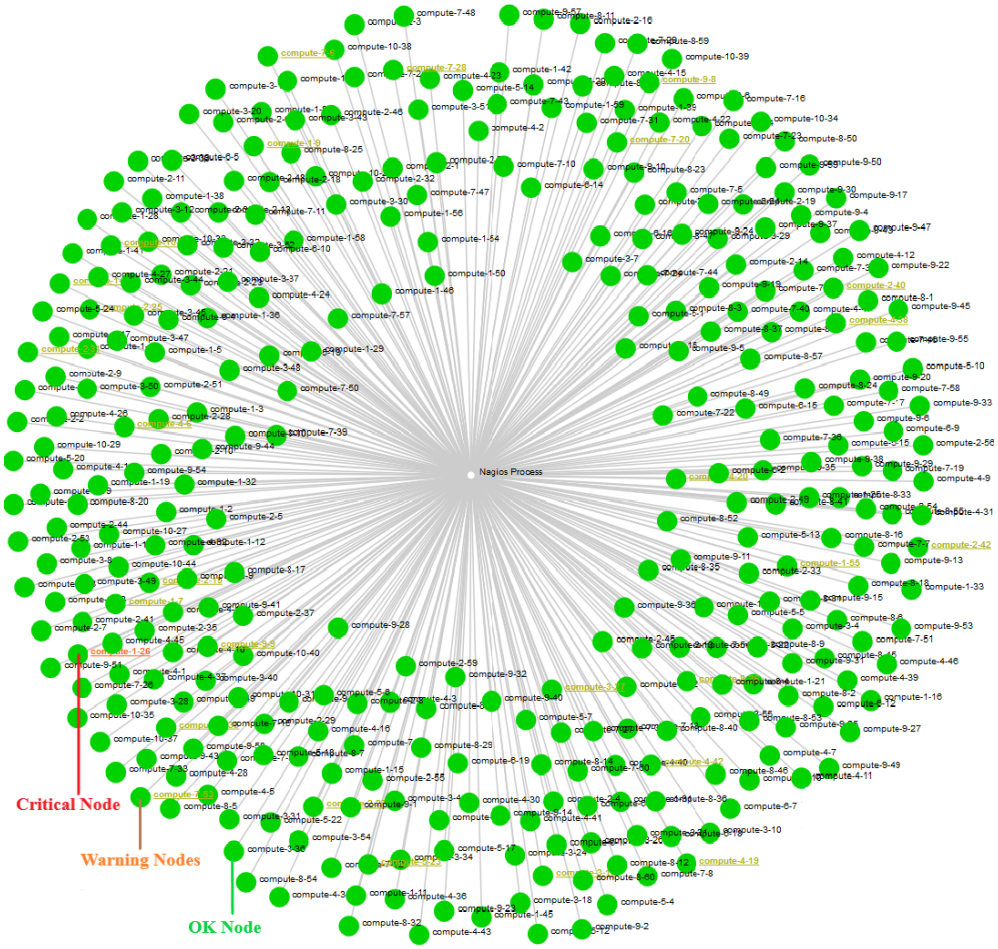


Fig. 13 Cluster Level Monitoring Visualization.

time to the HPC administrator to take appropriate action to prevent permanent damage to the CPU. This approach is more suitable for the visualization of a small number of metrics per host.

5.3.2 Visualization Using Parallel Coordinates

The parallel coordinates technique visualizes high-dimensional data and the relationship between dimensions. In parallel coordinates, the dimensions (or in this case monitored metrics) are displayed as parallel axes. The first vertical axis is for the time; the other vertical axes are in correlation order from left to right, meaning two adjacent axes have a stronger correlation. The collected data at a timestamp is illustrated as a curve of connected points corresponding to their axes' order and value. Fig. 15 visualizes the behaviors of different metrics over a particular duration. This

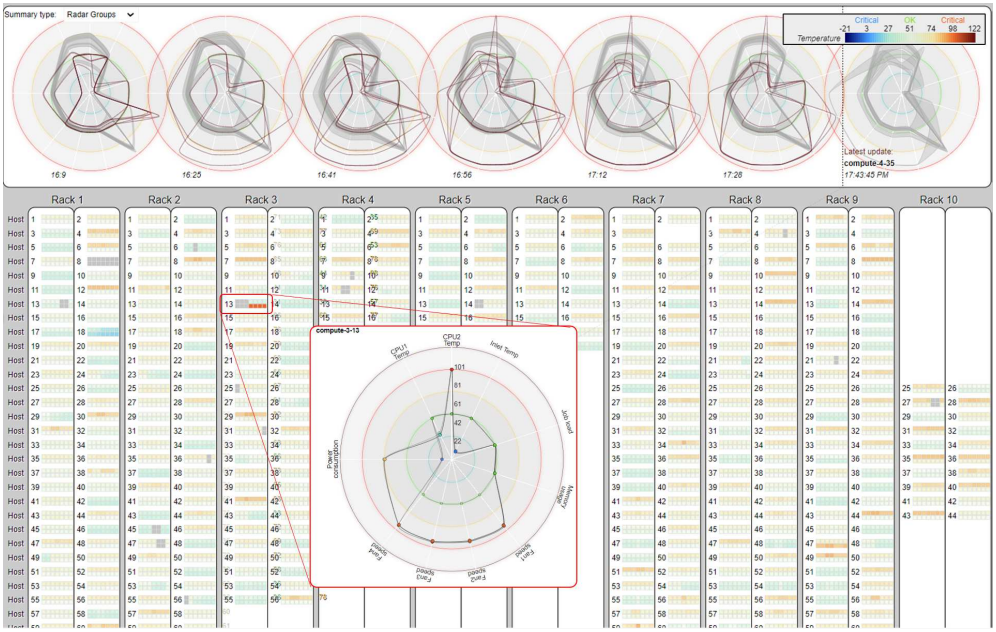


Fig. 14 Monitoring visualization using HiperView.

technique is effective for visualizing a comparatively smaller number of per-node metrics. Regardless, meaningful patterns like hidden metrics' density can be obscured in a cluster of lines when the number of metrics or data increases. However, with our tool, we mainly resolve the issue by adding the violin chart on each axis to display the density of the data. Additionally, we reduce the crossing problem between axes by ordering axes in correlation order. All axes can be re-arranged, disabled, or enabled by the user to reduce the visualization's complexity.

5.3.3 Visualization Using TimeRadar

HPC data centers require continuous monitoring of workload and system behaviors in terms of power consumption, thermal conditions, hardware resource consumption, and other metrics listed in Table 4. TimeRadar is an innovative visual analytics paradigm based on clustering and superimposing techniques to extract events of significance using these multivariate metrics [33]. Furthermore, these events are analyzed and characterized at each time step to determine the operating status of an HPC node. TimeRadar performs the following key tasks:

- Provides overview of the temporal event sequence for the nodes in the HPC data center.
- Computes the operating statuses of the nodes.
- Automates identification of anomalies such as sudden changes in the node behaviors.

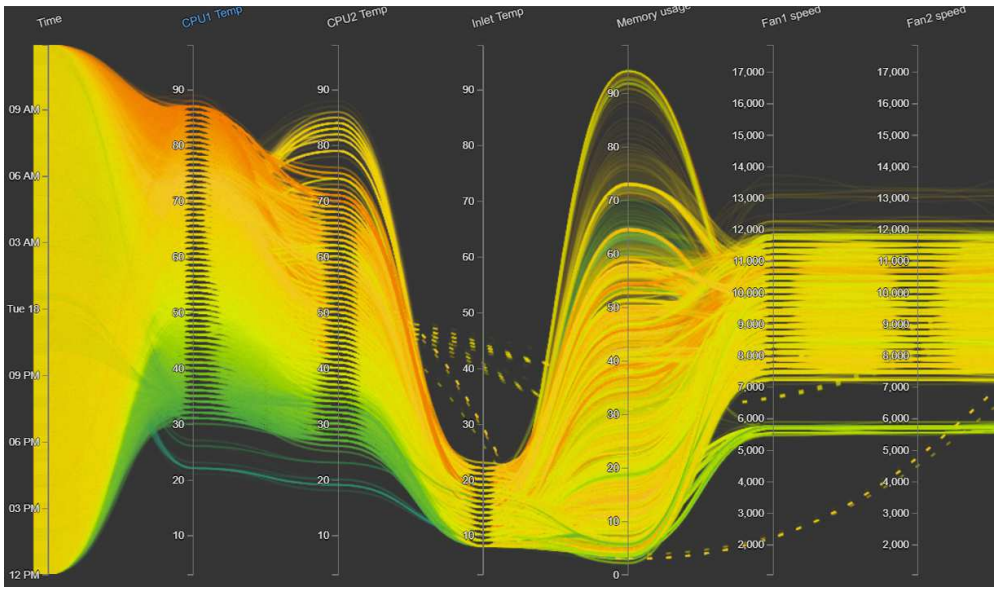


Fig. 15 HPC jobs monitoring using Parallel Coordinate technique.

- Updates the layout based on user input and node's behavioral changes (e.g., reordering events, running jobs, and grouping computing nodes with similar operating status).

Fig. 16 visualizes the dynamic behaviors of multiple events over a time period using TimeRadar. It shows the status of the jobs in terms of users, running jobs, number of nodes used in running a job, and power consumption by a user. Furthermore, it shows the correlation between jobs and events of significance incurred over different time steps. For example, some of the jobs running by `user5` (actual name is not used due to privacy concerns) are, comparatively, causing higher power consumption (i.e., 98 kWh) which is highlighted in the hosts' timeline accordingly.

6 Related Work

In this section, we compare the Redfish-Nagios tool with other studies from the following perspectives: scalability of the solution, the configuration of the tool, the need for an agent on the monitored node, communication mode (in-band or out-of-band), protocol to access BMC monitoring functions, and target monitoring scope. These comparisons are shown in Table 7.

Redfish is also used in some other studies to benchmark data centers. The study [39] compared different hardware monitoring mechanisms for data centers and showed that DMTF Redfish has a better scalability property than IPMI. The work [38] used Redfish to automate the Green500 methodology. The study [40] also used Redfish API in automating the dynamic thermal control of overheated CPUs in a high-performance

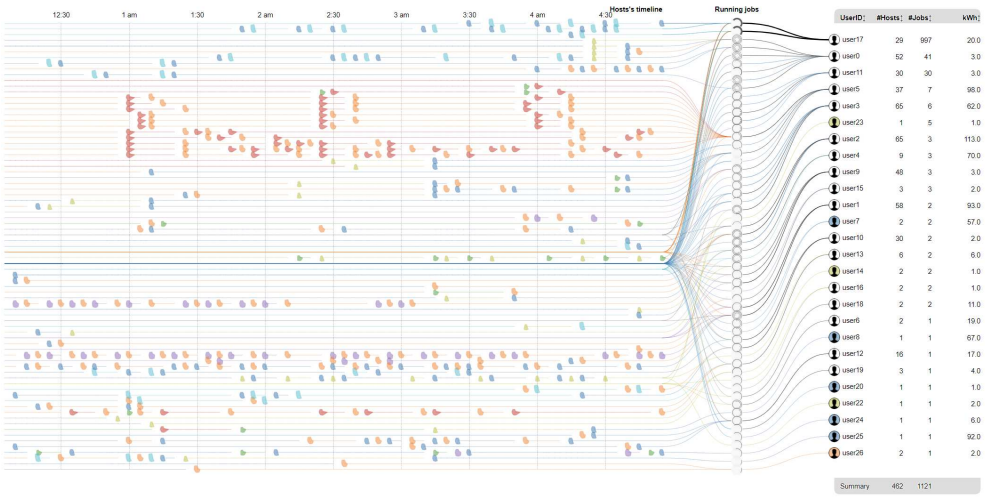


Fig. 16 HPC jobs monitoring using TimeRadar.

computing environment. In particular, [40] provided software-based component-level thermal control that is useful to handle the on-chip temperature. It allows the last-mile control of the thermal issues of the CPUs, not handled by the mechanical-level cooling solutions. Redfish provided a key role in collecting the CPU temperature in real time. Li et. al. [8] introduced *MonSTer* which is an "out-of-the-box" monitoring tool for HPC nodes. MonSTer used the Redfish API to collect different metrics from BMC, and resource management tools such as Univa Grid Engine (UGE) or Slurm to obtain workload information and resource usage data. MonSTer correlated workloads to utilization metrics and provides useful insights without having additional overhead on the workload and nodes. However, there is no research on utilizing and merging the DMTF Redfish technology with the main HPC monitoring tools, such as Nagios.

As described in Table 7, the Redfish-Nagios tool is the first study that enables Nagios to access BMC functions using Redfish API. In contrast to other studies, owing to Redfish technological enhancements, Redfish-Nagios provides improved scalability, automated configuration, agentless monitoring, and support for state-of-the-art out-of-band API to access BMC monitoring functions.

7 Conclusion and Future Work

The current Nagios monitoring tool is not efficient for modern data centers due to shortcomings originating from its in-band nature. These inadequacies arise from Nagios protocols including the requirement of monitoring specific in-band agents and plugins on the monitored nodes; the consumption of computational resources of the monitored node in executing the plugins, agents, or protocols; and the cumbersome manual configuration of the monitored nodes. We developed the Redfish-Nagios integration method, which enables Nagios to monitor HPC nodes and their components via BMC using state-of-the-art out-of-band Redfish API. The implemented method

Table 7 General comparison against other studies related to Nagios

Study	Scalability	Configuration	Agentless Monitoring	Communication Mode	BMC Protocol	Target
Nagios [34]	Limited	Manual	No	In-band	Not supported	Systems, networks, and infrastructure monitoring
Renita et al. [35]	Limited	Manual	No	In-band	Not supported	Network server monitoring
Luchian et al. [2]	Limited	Manual	No	In-band	Not supported	Cloud and network function virtualization (NFV) monitoring
Borghesi et al. [36]	Limited	Manual	No	In-band	Not supported	Anomaly detection in HPC systems
IPMI plugin for Nagios [37]	Limited	Manual	No	Out-of-band	IPMI	Systems, networks, and infrastructure monitoring
Hojati et al. [38]	Improved (using Redfish)	Automated	Yes	Out-of-band	Redfish	Energy efficiency
Redfish-Nagios (Our Work)	Improved (using Redfish)	Automated	Yes	Out-of-band	Redfish	Systems, networks, and infrastructure monitoring

removes the requirement of setting up any Nagios protocol, plugin, or agent. This integration saves important nodes' computational costs by shifting monitoring functions from the OS to the BMC. We also developed a Nagios configuration feature that automates configuration for monitoring the nodes and associated components and services on a large scale. In the future, we want to incorporate this capability into mainstream HPC management stacks (e.g., OpenHPC) so that the HPC community can benefit from the integration of Redfish with Nagios.

8 Declarations

8.1 Ethical Approval

Not applicable, because this article does not contain any studies with human or animal subjects.

8.2 Competing interests

The authors have no competing interests to declare that are relevant to the content of this article.

8.3 Authors' contributions

Conceptualization: All authors; Methodology: Ghazanfar Ali; Formal analysis and investigation: Ghazanfar Ali, Tommy Dang; Writing - original draft preparation: Ghazanfar; Writing - review and editing: All authors; Resources: Yong Chen, Alan Sill; Supervision: Yong Chen, Alan Sill, Jon Hass; Funding acquisition: Yong Chen, Jon Hass.

8.4 Funding

This research is supported in part by Dell Technologies and the National Science Foundation under grant CNS-1939140 (A U.S. National Science Foundation Industry-University Cooperative Research Center on Cloud and Autonomic Computing), OAC-1835892, and CNS-1817094.

8.5 Availability of data and materials

The proposed approach was tested using real-time telemetry data acquired at Texas Tech University (TTU)'s Quanah HPC cluster [11]. The related source codes are available as follows:

- The Redfish-Nagios integration-related code is available at: https://github.com/nsfcac/Nagios-Redfish-API-Integration.
- The Redfish metrics acquisition-related code is available at: https://github.com/nsfcac/DistributedMetricCollector.
- The web-based visualization-related artifacts are available at: https://idatavisualizationlab.github.io/HPCC.

Acknowledgments. This research is supported in part by Dell Technologies and the National Science Foundation under grant CNS-1939140 (A U.S. National Science Foundation Industry-University Cooperative Research Center on Cloud and Autonomic Computing), OAC-1835892, and CNS-1817094. We are also very grateful to the High Performance Computing Center of Texas Tech University for providing HPC resources for this project.

References

[1] Nagios: Nagios-The Industry Standard In IT Infrastructure Monitoring. https://www.nagios.org/

[2] Luchian, E., Docolin, P., Dobrota, V.: Advanced monitoring of the openstack nfv infrastructure: A nagios approach using snmp. 2016 12th IEEE International Symposium on Electronics and Telecommunications (ISETC) (2016). https://doi.org/10.1109/isetc.2016.7781055

[3] Ryder, T.: Nagios Core Administration Cookbook, 2nd edn. Packt Publishing Ltd, 35 Livery Street, Birmingham B3 2PB, UK (2016)

[4] Renita, J., Elizabeth, N.E.: Network's server monitoring and analysis using nagios. In: 2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), pp. 1904–1909 (2017). IEEE

[5] OpenHPC: OpenHPC Software Stack. https://openhpc.community/development/source-repository/

[6] Goncalves, G., *et al.*: A standard to rule them all: Redfish. IEEE Communications Standards Magazine **3**(2), 36–43 (2019)

[7] Hass, J.R.: Redfish facilities equipment management overview. In: Companion Proceedings of The10th International Conference on Utility and Cloud Computing, pp. 121–121 (2017)

[8] Li, J., Ali, G., Nguyen, N., Hass, J., Sill, A., Dang, T., Chen, Y.: Monster: An out-of-the-box monitoring tool for high performance computing systems. In: 2020 IEEE International Conference on Cluster Computing (CLUSTER), pp. 119–129 (2020). IEEE

[9] Hilland, J.: Redfish overview. In: Companion Proceedings of The10th International Conference on Utility and Cloud Computing, pp. 119–119 (2017)

[10] Ali, G., Hass, J., Sill, A., Hojati, E., Dang, T., Chen, Y.: Redfish-nagios: A scalable out-of-band data center monitoring framework based on redfish telemetry model. In: Fifth International Workshop on Systems and Network Telemetry and Analytics. SNTA '22, pp. 3–11. Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3526064.3534108. https://doi.org/10.1145/3526064.3534108

[11] HPCC: High Performance Computing Center. http:www.depts.ttu.edu/hpcc/

[12] Ghosh, S., Redekopp, M., *et al.*: Knightshift: Shifting the i/o burden in datacenters to management processor for energy efficiency. In: Computer Architecture, pp. 183–197. Springer, Berlin, Heidelberg (2012)

[13] Park, C., Joe, Y., Yoo, M., Lee, D., Kang, K.: Poster: Prototype of configurable redfish query proxy module. In: 2020 IEEE 28th International Conference on Network Protocols (ICNP), pp. 1–2 (2020). IEEE

[14] Suneja, S., *et al.*: Non-intrusive, out-of-band and out-of-the-box systems monitoring in the cloud. In: The 2014 ACM International Conference on Measurement and Modeling of Computer Systems. SIGMETRICS '14, pp. 249–261. ACM, New York, NY, USA (2014). https://doi.org/10.1145/2591971.2592009. http://doi.acm.org/10.1145/2591971.2592009

[15] Yoo, A.B., *et al.*: Slurm: Simple linux utility for resource management. In: Job Scheduling Strategies for Parallel Processing, pp. 44–60. Springer, Berlin, Heidelberg (2003)

[16] UGE: Univa Grid Engine. https://www.univa.com/

[17] Andrawos, M., Helmich, M.: Cloud Native Programming with Golang: Develop Microservice-based High Performance Web Apps for the Cloud with Go, 1st edn. Packt Publishing Ltd, 35 Livery Street, Birmingham B3 2PB, UK (2017)

[18] Treibig, J., Hager, G., Wellein, G.: Likwid: A lightweight performance-oriented tool suite for x86 multicore environments. In: 2010 39th International Conference on Parallel Processing Workshops, pp. 207–216 (2010). IEEE

[19] Hongsong, C., Xiaomei, W.: Design and implementation of cloud server remote management system based on impi protocol. In: 2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom), pp. 1475–1478 (2015). https://doi.org/10.1109/UIC-ATC-ScalCom-CBDCom-IoP.2015.266

[20] Rajachandrasekar, R., Besseron, X., Panda, D.K.: Monitoring and predicting hardware failures in hpc clusters with ftb-ipmi. In: 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum, pp. 1136–1143 (2012). https://doi.org/10.1109/IPDPSW.2012.139

[21] Zhang, S., *et al.*: Real time thermal management controller for data center. In: Fourteenth Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm), pp. 1346–1353 (2014). IEEE

[22] Bonkoski, A., *et al.*: Illuminating the security issues surrounding lights-out server management. In: Presented as Part of the 7th USENIX Workshop on Offensive Technologies. USENIX, Washington, D.C. (2013). https://www.usenix.org/conference/woot13/workshop-program/presentation/Bonkoski

[23] DMTF: DMTF's Redfish®. https://www.dmtf.org/standards/redfish

[24] Ali, G.: Nagios Redfish API Integration: Out-of-band (BMC) Based Monitoring. https://github.com/nsfcac/Nagios-Redfish-API-Integration

[25] Command Distributor. https://exchange.nagios.org/directory/Addons/External-Commands/CmdDist/details

[26] Check Nagios Summary. https://exchange.nagios.org/directory/Plugins/Network-and-Systems-Management/Nagios/check_nagios_summary/details

[27] Check Remote Nagios. https://exchange.nagios.org/directory/Addons/Distributed-Monitoring/check_remotenagios/details

[28] Check Summary. https://exchange.nagios.org/directory/Plugins/Network-and-Systems-Management/Nagios/check_summary/details

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

[30] Kim, J., *et al.*: Performance evaluation of multithreaded computations for cpu bounded task. In: 2016 International Conference on Platform Technology and Service (PlatCon), pp. 1–5 (2016). https://doi.org/10.1109/PlatCon.2016.7456816

[31] Rinku, D.R., Asha Rani, M.: Analysis of multi-threading time metric on single and multi-core cpus with matrix multiplication. In: 2017 Third International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB), pp. 152–155 (2017). https://doi.org/10.1109/AEEICB.2017.7972402

[32] Dang, T., Nguyen, N., Chen, Y.: Hiperview: real-time monitoring of dynamic behaviors of high-performance computing centers. The Journal of Supercomputing **77** (2021). https://doi.org/10.1007/s11227-021-03724-5

[33] Nguyen, N.V., Hass, J., Dang, T.: Timeradar: Visualizing the dynamics of multivariate communities via timeline views. In: 2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC), pp. 350–356 (2021). IEEE

[34] Enterprises, N.: Nagios (2017)

[35] Renita, J., *et al.*: Network's server monitoring and analysis using nagios. In: 2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), pp. 1904–1909 (2017). https://doi.org/10.1109/WiSPNET.2017.8300092

[36] Borghesi, A., *et al.*: Anomaly detection and anticipation in high performance computing systems. IEEE Transactions on Parallel and Distributed Systems **33**(4), 739–750 (2022). https://doi.org/10.1109/TPDS.2021.3082802

[37] Thomas-Krenn.AG: IPMI Sensor Monitoring Plugin. Thomas-Krenn.AG (2020).

https://www.thomas-krenn.com/en/wiki/IPMI_Sensor_Monitoring_Plugin_setup

[38] Hojati, E., *et al.*: Redfish green500 benchmarker (rgb): Towards automation of the green500 process for data centers. In: 2020 IEEE Green Technologies Conference(GreenTech), pp. 47–52 (2020). https://doi.org/10.1109/GreenTech46478.2020.9289729

[39] Hojati, E., *et al.*: Benchmarking automated hardware management technologies for modern data centers and cloud environments. In: Proceedings of The10th International Conference on Utility and Cloud Computing, pp. 195–196 (2017)

[40] Ali, G., Wofford, L., Turner, C., Chen, Y.: Automating cpu dynamic thermal control for high performance computing. In: 2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid), pp. 514–523 (2022). IEEE