Redfish-Nagios: A Scalable Out-of-Band Data Center Monitoring Framework Based on Redfish Telemetry Model

Ghazanfar Ali
Texas Tech University
Lubbock, USA
ghazanfar.ali@ttu.edu

Jon Hass
Dell Technologies
Austin, USA
jon.hass@dell.com

Alan Sill
Texas Tech University
Lubbock, USA
alan.sill@ttu.edu

Elham Hojati
Texas Tech University
Lubbock, USA
elham.hojati@ttu.edu

Tommy Dang
Texas Tech University
Lubbock, USA
tommy.dang@ttu.edu

Yong Chen
Texas Tech University
Lubbock, USA
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 infrastructure of modern data centers. Nagios is one of the widely used industry-standard tools for data center infrastructure monitoring, which mainly include monitoring of nodes and associated hardware and software components. However, current Nagios monitoring has special requirements that introduce several limitations. First, a 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 the computational cost by removing the requirements of running complex Nagios-native protocols and agents on the monitored nodes. The Redfish-Nagios integration methodology enables monitoring of next-generation HPC systems using the scalable and modern Redfish telemetry model and interface.

CCS CONCEPTS

• **General and reference → Metrics**; **Measurement**; **Evaluation**; **Empirical studies**.

SNTA '22, June 30, 2022, Minneapolis, MN, USA

ACM ISBN 978-1-4503-9315-7/22/06.
https://doi.org/10.1145/3526064.3534108

KEYWORDS

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

ACM Reference Format:
Ghazanfar Ali, Jon Hass, Alan Sill, Elham Hojati, Tommy Dang, and Yong Chen. 2022. Redfish-Nagios: A Scalable Out-of-Band Data Center Monitoring Framework Based on Redfish Telemetry Model. In *Proceedings of the Fifth Int'l Workshop on Systems and Network Telemetry and Analytics (SNTA '22), June 30, 2022, Minneapolis, MN, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3526064.3534108

1 INTRODUCTION

As data center technologies expand, the need to improve the 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 [18] [17] [25] [23] 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 [19]. 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 [8] [9] [16] [10]. 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 integration of Redfish with Nagios are described below.

1.1 Motivations and Contributions

To our knowledge, this research is the first study that investigates and integrates Redfish with the Nagios Core. Redfish integration

with Nagios is potentially a paradigm shift for Nagios-based monitoring. The key contributions of this study include enabling agent-less monitoring, automating configuration, and eliminating Nagios in-band components.

1.1.1 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 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.

1.1.2 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.

1.1.3 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.

1.2 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 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 [31] or Univa grid engine (UGE) [30] 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 an 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 consumption 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.

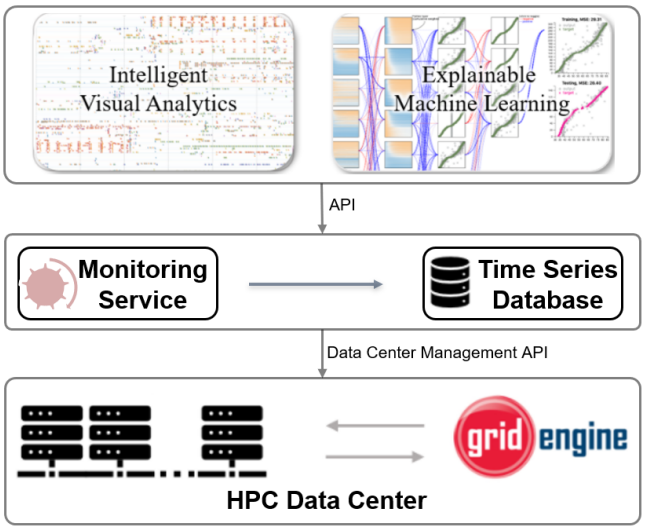

Figure 1: Overview of HPC data center monitoring framework

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 remote monitored nodes via the operating system to acquire telemetry data. This mechanism not only complicates 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 [2] and leveraged the LIKWID HPC performance tool [29]. 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 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.

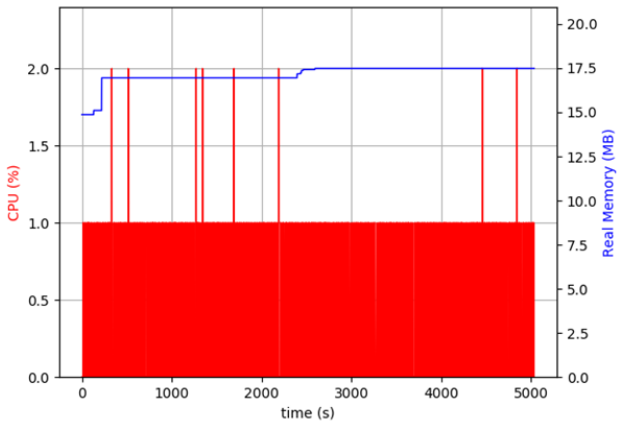

Figure 2: Computational resources overhead of in-band monitoring

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). Study[7] showed that offloading monitoring tasks from the node's operating system to the BMC reduced the power consumption by a factor of 2.6. Significant progress has been made in the arena of BMC hardware and software, including Redfish and the Redfish telemetry model [8–10, 16]. There is an increasing trend to perform system monitoring using OOB mechanisms [20, 26]. 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 [13, 21], 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[32]. 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[3], scalability, and complexity due to being a bit-wise protocol.

2.3.2 Redfish. Redfish [5] [8] [9] [16] [10] 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. 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. Chassis also can include other chassis and consists of a variety of sensors and fans. 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.

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.3) 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.4) to

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

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.3. As depicted in Fig.3, the protocol-specific in-band endpoint and monitored entity-specific monitoring agent are 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.

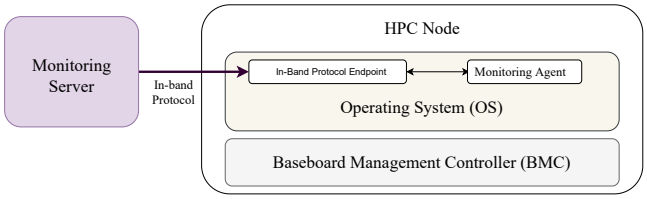

Figure 3: In-band communication mechanism

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.4. As depicted in Fig.4, 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. Redfish API, a state-of-the-art OOB-based model and standard, is used to achieve these objectives.

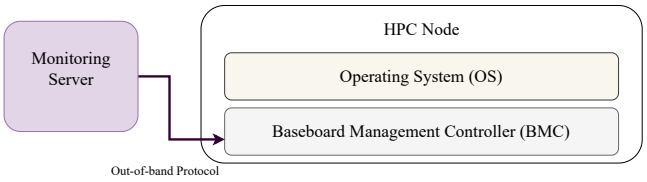

Figure 4: OOB communication mechanism

3.3 Redfish-Nagios Tool Architecture

Fig. 5 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

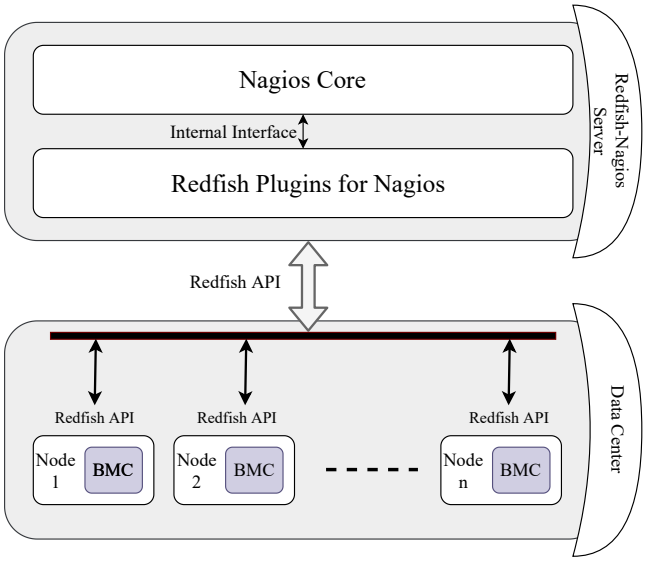

Figure 5: Redfish-Nagios Tool Architecture

of the node supports Redfish. The Nagios Core communicates to 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 a standardized, efficient, generic, and automated Nagios-based data center monitoring. The Redfish-Nagios tool supports six Redfish-based plugins for Nagios as listed in Table 2. The above plugins use the following Redfish

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 CPU temperature
check_fans	Acquires fan health and speed
check_temperature	Acquires CPU temperature

URIs to acquire different metrics, as shown in Table 3.

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 a 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 4.

When the monitoring data is a numeric value, the value is translated to one of three possible Nagios states as shown in Table 4 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 a 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, a HPC administrator might unintentionally miss a high

Table 4: 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

Table 5: 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

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 detailed source code for the Redfish plugins is available on GitHub [1].

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 [14] 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) [27], which implements the Redfish API [5] 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 5 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 are 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.

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 1 and listing 2, respectively.

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

Listing 1: Node configuration information

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

Listing 2: 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 [15], the monitoring workload is distributed evenly among the available cores [24]. 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. 6 shows 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.

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 [14]. The results presented in this section contain the performance of the Redfish-Nagios server and monitoring visualization using the Redfish-Nagios tool.

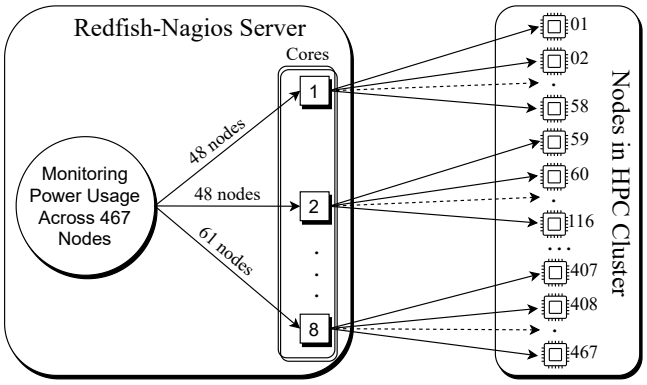

Figure 6: Parallelization of Redfish requests in Redfish-Nagios Server, in monitoring power usage across 467 nodes in the Quanah cluster.

5.1 Redfish-Nagios Server Performance

Our Redfish-Nagios server ran 2,802 monitoring checks (467 nodes X 6 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.

5.2 Monitoring Visualization Using the Redfish-Nagios Tool

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

5.2.1 Component Level: Fig. 7 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.

5.2.2 Node Level: Fig. 8 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.

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

Figure 7: Node's Component Level Monitoring Visualization

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!

Figure 8: Node Level Monitoring Visualization.

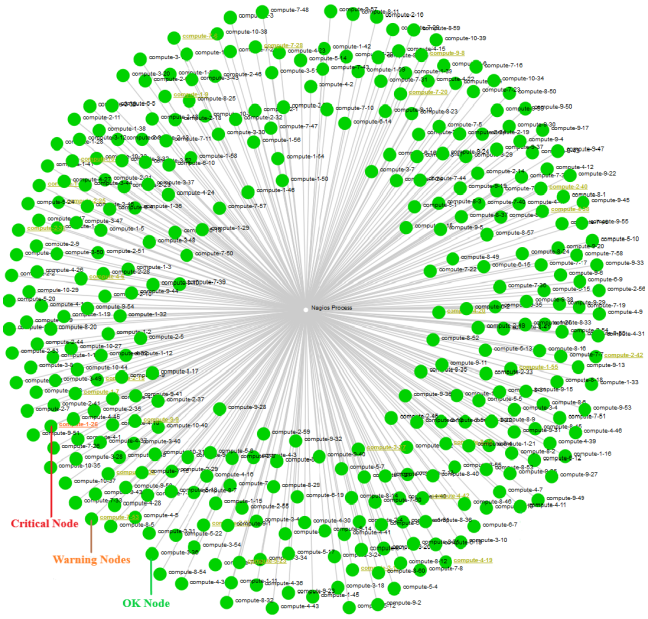

Figure 9: Cluster Level Monitoring Visualization.

5.2.3 Cluster Level: Node-level visualization is not sufficient to provide a high-level summary of the large-scale cluster. Fig. 9 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.

6 RELATED WORK

In this section, we compare the Redfish-Nagios tool with other studies from the following perspectives: scalability of the solution, configuration of the tool, need of 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 6.

Redfish is also used in some other recent studies to benchmark data centers. The study [11] compared different hardware monitoring mechanisms for data centers and showed that DMTF Redfish has a better scalability property than IPMI. The work [12] used Redfish to automate Green500 methodology. However, there is no research on utilizing and merging the DMTF Redfish technology with the main HPC monitoring tools, such as Nagios.

Table 6: General comparison against other studies related to Nagios

Study	Scalability	Configuration	Agentless Monitoring	Communication Mode	BMC Protocol	Target
Nagios [6]	Limited	Manual	No	In-band	Not supported	Systems, networks, and infrastructure monitoring
Renita et al. [22]	Limited	Manual	No	In-band	Not supported	Network server monitoring
Luchian et al. [17]	Limited	Manual	No	In-band	Not supported	Cloud and network function virtualization (NFV) monitoring
Borghesi et al. [4]	Limited	Manual	No	In-band	Not supported	Anomaly detection in HPC systems
IPMI plugin for Nagios [28]	Limited	Manual	No	Out-of-band	IPMI	Systems, networks, and infrastructure monitoring
Hojati et al. [12]	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

As described in Table 6, 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 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.

ACKNOWLEDGMENT

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) and OAC-1835892. We are also very grateful to the High Performance Computing Center of Texas Tech University for providing HPC resources for this project.

REFERENCES

[1] Ghazanfar Ali. 2020. *Nagios Redfish API Integration: Out-of-band (BMC) based Monitoring.* Retrieved May, 2022 from https://github.com/nsfcac/Nagios-Redfish-API-Integration
[2] Mina Andrawos and Martin Helmich. 2017. *Cloud Native Programming with Golang: Develop microservice-based high performance web apps for the cloud with Go.* Packt Publishing Ltd.
[3] Anthony Bonkoski et al. 2013. 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. https://www.usenix.org/conference/woot13/workshop-program/presentation/Bonkoski
[4] A. Borghesi et al. 2022. Anomaly Detection and Anticipation in High Performance Computing Systems. *IEEE Transactions on Parallel and Distributed Systems* 33, 4 (2022), 739–750. https://doi.org/10.1109/TPDS.2021.3082802
[5] DMTF. 2020. *DMTF's Redfish®.* Retrieved May, 2020 from https://www.dmtf.org/standards/redfish
[6] Nagios Enterprises. 2017. Nagios.
[7] Sabyasachi Ghosh, Mark Redekopp, et al. 2012. KnightShift: Shifting the I/O Burden in Datacenters to Management Processor for Energy Efficiency. In *Computer Architecture.* Springer Berlin Heidelberg, Berlin, Heidelberg, 183–197.
[8] Glauco Goncalves et al. 2019. A standard to rule them all: Redfish. *IEEE Communications Standards Magazine* 3, 2 (2019), 36–43.
[9] Jon R Hass. 2017. Redfish Facilities Equipment Management Overview. In *Companion Proceedings of the10th International Conference on Utility and Cloud Computing.* 121–121.
[10] Jeff Hilland. 2017. Redfish Overview. In *Companion Proceedings of the10th International Conference on Utility and Cloud Computing.* 119–119.
[11] Elham Hojati et al. 2017. Benchmarking automated hardware management technologies for modern data centers and cloud environments. In *Proceedings of the10th International Conference on Utility and Cloud Computing.* 195–196.
[12] E. Hojati et al. 2020. Redfish Green500 Benchmarker (RGB): Towards Automation of the Green500 Process for Data Centers. In *2020 IEEE Green Technologies Conference(GreenTech).* 47–52. https://doi.org/10.1109/GreenTech46478.2020.9289729
[13] C. Hongsong and W. Xiaomei. 2015. 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).*

1475–1478. https://doi.org/10.1109/UIC-ATC-ScalCom-CBDCom-IoP.2015.266

[14] HPCC. 2022. *High Performance Computing Center.* Retrieved February, 2022 from http:www.depts.ttu.edu/hpcc/

[15] J. Kim et al. 2016. Performance Evaluation of Multithreaded Computations for CPU Bounded Task. In *2016 International Conference on Platform Technology and Service (PlatCon)*. 1–5. https://doi.org/10.1109/PlatCon.2016.7456816

[16] Jie Li, Ghazanfar Ali, Ngan Nguyen, Jon Hass, Alan Sill, Tommy Dang, and Yang Chen. 2020. MonSTer: An Out-of-the-Box Monitoring Tool for High Performance Computing Systems. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 119–129.

[17] Eduard Luchian, Paul Docolin, and Virgil Dobrota. 2016. 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

[18] Nagios. 2020. *Nagios-The Industry Standard In IT Infrastructure Monitoring.* Retrieved May, 2020 from https://www.nagios.org/

[19] OpenHPC. 2020. *OpenHPC Software Stack.* Retrieved December, 2020 from https://openhpc.community/development/source-repository/

[20] Chanyoung Park, Yoonsue Joe, Myounghwan Yoo, Dongeun Lee, and Kyungtae Kang. 2020. Poster: Prototype of Configurable Redfish Query Proxy Module. In *2020 IEEE 28th International Conference on Network Protocols (ICNP)*. IEEE, 1–2.

[21] R. Rajachandrasekar, X. Besseron, and D. K. Panda. 2012. Monitoring and Predicting Hardware Failures in HPC Clusters with FTB-IPMI. In *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum*. 1136–1143. https://doi.org/10.1109/IPDPSW.2012.139

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

[23] J Renita and N Edna Elizabeth. 2017. Network's server monitoring and analysis using Nagios. In *2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*. IEEE, 1904–1909.

[24] D. R. Rinku and M. Asha Rani. 2017. 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)*. 152–155. https://doi.org/10.1109/AEEICB.2017.7972402

[25] Tom Ryder. 2016. *Nagios core administration cookbook.* Packt Publishing Ltd.

[26] Sahil Suneja et al. 2014. 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* (Austin, Texas, USA) *(SIGMETRICS '14)*. ACM, New York, NY, USA, 249–261. https://doi.org/10.1145/2591971.2592009

[27] DELL Technologies. 2020. *Integrated Dell Remote Access Controller (iDRAC).* Retrieved May, 2020 from https://www.delltechnologies.com/en-us/solutions/openmanage/idrac.htm

[28] Thomas-Krenn.AG. 2020. IPMI Sensor Monitoring Plugin. https://www.thomas-krenn.com/en/wiki/IPMI_Sensor_Monitoring_Plugin_setup

[29] Jan Treibig, Georg Hager, and Gerhard Wellein. 2010. Likwid: A lightweight performance-oriented tool suite for x86 multicore environments. In *2010 39th International Conference on Parallel Processing Workshops*. IEEE, 207–216.

[30] UGE. 2020. *Univa Grid Engine.* Retrieved May, 2020 from https://www.univa.com/

[31] Andy B. Yoo et al. 2003. SLURM: Simple Linux Utility for Resource Management. In *Job Scheduling Strategies for Parallel Processing*. Springer Berlin Heidelberg, Berlin, Heidelberg, 44–60.

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