University of Victoria
Engineering & Computer Science Co-op
Work Term Report
Fall 2021

Monitoring and Anomaly Detection for Cloud Computing Infrastructure

High Energy Physics Research Computing Group
Department of Physics, University of Victoria
Victoria, BC, Canada

Victor Kamel
V00963333
W-1
Computer Science
vkamel@uvic.ca
December 17, 2021

In partial fulfillment of the academic requirements of this co-op term

Supervisor's Approval: To be completed by Co-op Employer

This report will be handled by UVic Co-op staff and will be read by one assigned report marker who may be a co-op staff member within the Engineering and Computer Science Co-operative Education Program, or a UVic faculty member or teaching assistant. The report will be retained and available to the student or, subject to the student's right to appeal a grade, held for one year after which it will be deleted.

I approve the release of this report to the University of Victoria for evaluation purposes only.

Signature: Colson Driemel Position: Research Assistant Date: Dec 16th 2021

Name (print): Colson Driemel E-Mail: colsond@uvic.ca

For (Company Name) University of Victoria

Executive Summary

The High Energy Physics Research Computing (HEPRC) group at the University of Victoria manages computational clouds for high energy physics applications. This involves maintaining a support infrastructure of machines that must maintain a high degree of availability. To achieve this, a monitoring solution must be put in place to capture performance metrics from running machines to ensure their consistent and correct operation over time. In addition, this solution should be flexible enough to work with any of the machines in the supporting infrastructure, the cloud hypervisors, and the worker nodes running on the clouds. In this report, a number of different software packages and possible configurations are tested and evaluated. An appropriate approach that integrates Ganglia and Nagios, along with custom software written for this application is implemented within the HEPRC infrastructure. Furthermore, the time series data generated by the monitoring system is then used to train an anomaly detection model, which is applied to detect when machines do not behave as they should and alert system administrators. This system can make use of multiple metrics to detect more complex anomalies that the threshold-based single metric alerts performed by Nagios cannot.

List of Figures and Tables

Figures

Tables

Glossary

API	*Application Programming Interface.* Enables intercommunication between software.
cloud	A collection of computational resources available on-demand over the internet.
cron	A Linux utility that schedules processes to run periodically.
daemon	A background process on a Linux machine.
host	A computer connected to a computer network.
HTTP	*Hypertext Transfer Protocol.* A protocol for transmitting resources over the internet.
IT	*Information Technology.* Use of computers for the manipulation of information, typically in a business context.
metric	A measurable performance characteristic of a machine such as memory or network usage.
open source	Software whose source code is freely available.
private subnet	Hosts in a private subnet only need a private IP address may not be directly reachable from the internet.
SQL	*Structured Query Language.* A language used for managing data in relational databases.
SSH	*Secure Shell.* A secure remote communication protocol.
TCP	*Transmission Control Protocol.* A communication protocol that re-transmits lost packets.
UDP	*User Datagram Protocol.* A communication protocol that does not re-transmit lost packets.
XDR	*External Data Representation.* A data serialization format for transmitting data over the network.

Table of Contents

1 Introduction

The High Energy Physics Research Computing Group (HEPRC) is a research group within the University of Victoria's Physics and Astronomy Department that manages cloud computational resources for high energy physics projects such as the ATLAS [1] experiment at CERN and Belle II [2] at the KEK Laboratory in Japan. In particular, their `cloudscheduler` [3] software automatically provisions virtual machines on private and public clouds to run computational workloads specified by the projects. In their daily operations, the HEPRC group manages a significant number of computer instances—both physical and virtual—with bespoke configurations.

1.1 Problem definition

Although the HEPRC group has monitoring functionality in place for the individual systems within their infrastructure, which makes it possible to detect failures in those components, the separation of these metrics makes it difficult to detect smaller errors that cannot be readily observed when looking at these systems in isolation. A centralized monitoring solution that can be deployed to all machines in their infrastructure is therefore proposed and implemented in order to make it easier to visualize and compare the metrics in one central location. In addition, an anomaly detection system is created in order to automate the detection of errors that are currently detected manually and to leverage machine learning approaches in order to find hidden patterns that are difficult for system administrators to detect in the data.

1.2 Requirements

The most important characteristics that a monitoring solution should possess were identified to be the ability to:

1. Be deployed to any of the machines in the HEPRC infrastructure
2. Make the following data available to administrators:
   (a) System metrics and performance data in a graphable time series format
   (b) State information on key processes
   (c) Security information such as the number of active remote connections

3. Detect anomalous behaviour and alert administrators when their attention is required
4. Display graphs of time series data in the web interface
5. Allow the collection of custom metrics
6. Allow the dynamic adding and removal of monitored hosts
7. Scale as the monitoring needs of the infrastructure grow.

Another design consideration was the method in which metrics are collected over the network. Since some machines may be part of a private subnet that the monitoring server cannot access directly, it was decided that the monitored hosts must send metric data to the server without the server needing to explicitly establish a connection to the monitored hosts. This shall be referred to as *passive* data collection.

2 Monitoring Infrastructure

As the number of machines that are required to be functional in an infrastructure increases, ensuring their correct and consistent operation becomes difficult. To that end, organizations like CERN employ software to collect and report operational data from running machines to a centralized location where it may be visualized for system administrators [4]. Commonly, such systems also have provisions for detecting and reporting anomalous behavior to administrators so that they can investigate the source of the problem and undertake corrective action if required.

2.1 Options considered

Due to the importance of capable monitoring systems to organizations that utilize many machines, there exist a multitude of different software packages designed to solve this problem. However, many of them do not fully meet the requirements defined in §1.2 and are thus not ideal for the HEPRC infrastructure. As such, a candidate set of software was identified for preliminary consideration during the initial research phase.

2.1.1 Cacti

Cacti [5] is a monitoring framework that uses RRDtool [6] and the MariaDB [7] relational database for time series data storage and graphing. It is based on the LAMP (Linux, Apache, MySQL & PHP) [8, 9, 10] web

server stack and is commonly used to monitor network traffic [11]. Cacti also provides a high performance multi-threaded data collector, Spine, written in C, that can scale to a larger number of hosts then the default `cmd.php` poller [12]. Cacti also has the ability to send alerts based on thresholds using the `thold` [13] plugin. However, Cacti is primarily designed to collect data by polling the monitored machines directly, which is not ideal for the HEPRC's application.

2.1.2 Munin

Similar to Cacti, Munin [14] is also a network resource monitoring tool that functions as a frontend to RRDtool. It is written in Perl [15] and is "designed to be very plug and play" [14]. Munin collects data by connecting to an agent, `munin-node`, installed on the monitored host at set intervals [16]. Additionally, Munin has the ability to send alerts, or integrate with software such as Nagios (evaluated in §2.1.7) for this functionality. However, since the Munin server must connect to the monitored node by default, this solution does not fulfill the requirements.

2.1.3 Prometheus

Prometheus [17] is a monitoring system built by SoundCloud in 2012, which has since become an independent open source project [18]. It is written in Go [19], and designed to work well with purely numeric time series data and to value reliability over accuracy. Metrics are gathered by polling *exporters* installed on the machines, and Prometheus functions as the database [20]. The data can then be visualized by the Grafana [21] dashboard software. Alerts can also be triggered in Prometheus and sent using Alertmanager [22]. Since Prometheus is exclusively designed for numeric data and uses an *active* model of data collection, it does not fit the requirements.

2.1.4 ELK Stack

The Elasticsearch, Logstash and Kibana (also known as the Elastic) stack [23] is a collection of software used primarily for log management and analytics [24]. Logstash ingests and transforms data from multiple sources, Elasticsearch stores and indexes the data and Kibana queries Elasticsearch to visualize the data [23]. Logging numeric time series data is possible with Metricbeat [25], a data shipper that can be used

to send data to Logstash for processing. It can also send data to Elasticsearch directly, which reduces the required performance overhead. Alerting can be set up using Kibana [26]. However, this application does not leverage the full power of Elasticsearch, since it is primarily built as full-text document search engine rather than a time series database.

2.1.5 Zabbix

Zabbix [27] is a system monitoring tool for IT components that stores data in a MariaDB SQL database. Zabbix has a more modern web interface compared to the other options considered, and provides a "batteries included" monitoring experience. Zabbix supports graphing time series data, notifications and host auto-discovery without the need for any other software [28]. Data is collected from the machines using the `Zabbix Agent`, which supports both *active* and *passive* data transmission[1] [29]. Zabbix is completely configured from the web interface and does not require any manual editing of config files. However, due to all of this integrated functionality, the Zabbix server may run slower than the alternatives. Since it appears to fit all of the requirements, Zabbix was one of the solutions selected for further testing.

2.1.6 Ganglia

Ganglia [30] is grid monitoring solution with robust support for gathering metrics at scale. Data is gathered using the resident `gmond` agent that is installed on the hosts. The `gmond` instance will send data to a `gmetad` instance, which then stores and graphs the data using the RRDtool time series database. The data can then be displayed using the `ganglia-web` (or `gweb`) PHP-based web interface. Multiple host instances can be arranged into a *grids*, and multiple grids can then be arranged into *clusters*. The Ganglia monitoring daemon `gmond` is available in the CentOS and many other Linux distributions' repositories, which makes it very easy to install. Custom metrics can be added using `gmetric`, but many of the most important metrics are available by default without any configuration required. It is dynamic, and allows the adding or removal of hosts without any additional configuration on the server. These aspects make Ganglia ideal for the HEPRC's infrastructure. Although Ganglia has no built-in support for notifications or alerts of any kind, it can integrate with other software such as Nagios that can provide this functionality.

[1] Zabbix defines "active" and "passive" from the perspective of the agent rather than the server, so "active" refers to what was defined as *passive* data collection in §1.2.

2.1.7 Nagios

Nagios Core [31] is an open source monitoring and alerting system for IT infrastructure written in C. Nagios only provides alerting functionality by default, but is built to incorporate *plugins* that allow it to collect data from monitoring sources. Nagios performs checks to determine the state of the *objects* that it is monitoring such as hosts and services. These checks can be *active*, where Nagios performs the check by itself, or *passive*, where it processes checks submitted by external applications [32, 33]. Depending on the result of the check, Nagios will schedule the next check and alert an administrator if there is an error. Although Nagios does not have any way to store time series data or create graphs, plugins such as PNP4Nagios [34] enable this functionality by storing data in RRD files like many of the other solutions considered do. Although Nagios must be aware of all of the hosts that it is monitoring, which makes difficult to add and remove hosts dynamically, this can be solved by integrating it with a dynamic system such as Ganglia.

2.2 Possible configurations

Out of the many options under consideration, Zabbix, Ganglia, and Nagios were selected for further testing as they were the best fit for the requirements. While Ganglia and Nagios do not fully meet the criteria on their own, it is possible to integrate them together so that they cover each other's shortcomings.

In order to verify these software options in relation to the HEPRC group's requirements, several different configurations were created in an OpenStack [35] environment and subsequently evaluated. These systems were configured to monitor the test set of metrics and alerts outlined in Table 1.

Table 1: Test metrics and alerts.

Time Series with Graphs	**Alerts / Reports**
RAM usage	Swap usage $>$ 50%
Swap usage	Disk usage $>$ 90%
CPU usage (%)	SSH login from an unrecognized network
System load	Service httpd not running
Disk usage	Service sshd not running
Network bytes (in/out)	
Network packages (in/out)	

2.2.1 Zabbix Standalone

Testing Zabbix revealed that the Zabbix Agent that was installed onto the monitored hosts used more system resources than the agents for other options available, which is not ideal for many of the machines in the HEPRC infrastructure, especially the cloud worker nodes. In addition, configuring the Zabbix server to fit the HEPRC's needs was more difficult than the other options considered, because the flexibility allowed by the web interface was limited. The server itself was slower as well. Finally, this solution requires a separate SQL database to be installed and running at all times, while the other solutions use RRDtool, which does not need to be running constantly in the background as a daemon.

2.2.2 Nagios with PNP4Nagios

Nagios was tested with the PNP4Nagios plugin for manipulating time series data. Although this solution allows Nagios to be used standalone, there is no easily-deployable agent with a small performance footprint that could be used for this application. To make this system functional, a monitoring agent that sends data using the NCSA plugin would likely need to be written from scratch. Host management would also be difficult since Nagios ignores service checks for hosts that it does not know of beforehand.

2.2.3 Nagios with Ganglia Independent

Testing Ganglia independently from Nagios covers the shortcomings of both pieces of software, however it requires two different agents to be installed on each machine since Nagios cannot use the data collected by Ganglia. This also does not mitigate the issue with creating the agent that can send data passively to the monitoring server, nor the issue of adding hosts dynamically.

2.2.4 Nagios with Ganglia Integration

In this configuration, Ganglia is used exclusively to collect and display the data, which makes it simple to deploy the agent to the machines. It then makes this data available to Nagios. This way, Nagios can actively check the Ganglia server for the metrics that it requires, and send out notifications if something goes wrong. Although scripts to support this sort of integration come packaged with the `gweb`, additional custom scripts

may be required depending on the application. This combination makes the most practical sense because only one agent is needed to be installed on host systems to enable monitoring.

2.3 Final system design

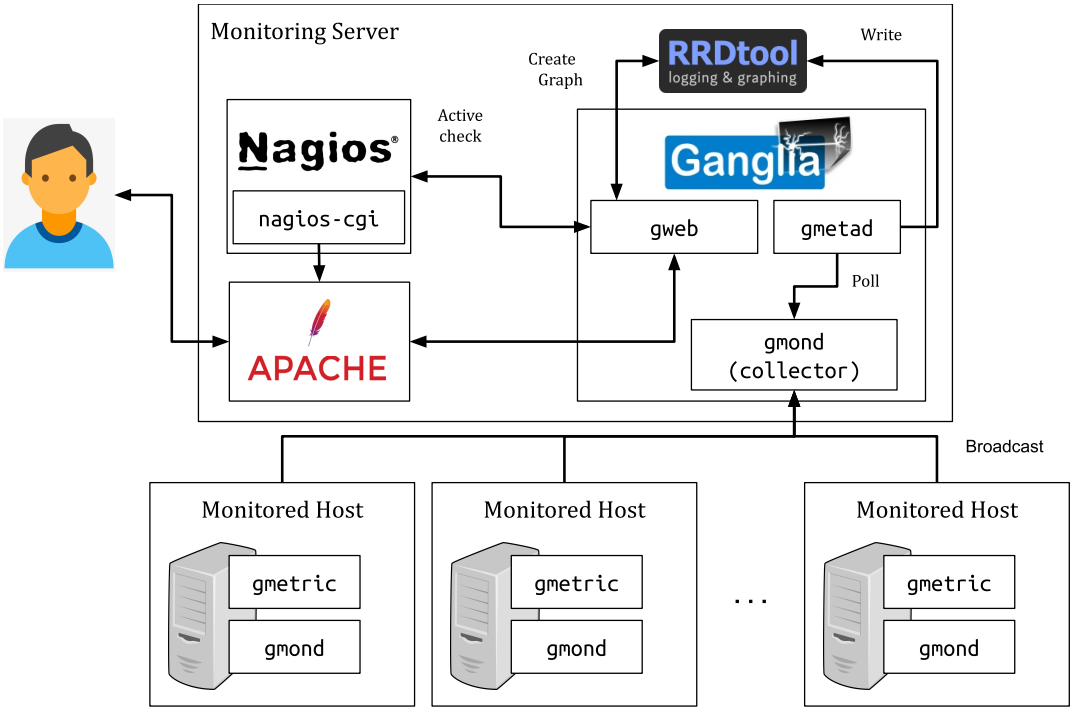

Figure 1: Diagram of monitoring system.

In the finalized system, shown in Figure 1, Ganglia's `gmond` and `gmetric` monitoring agents are deployed to hosts that require monitoring. Metric information is thus sent to a collector `gmond` instance through XDR over UDP. The `gmetad` service on the monitoring server will then poll the collector `gmond` over TCP to gather the metrics and write them to Round Robin Database (RRD) files on the monitoring server. When the Ganglia web interface requires graphs to be generated, it requests them from RRDtool. Users can then visualize the data in the web interface as shown in Figure 2, and Nagios can access the current metric information from `gweb`. If Nagios detects a metric with a value that is over the predefined threshold, it will alert the user by sending an email. The services that Nagios is monitoring can also be viewed in its web interface as seen in Figure 3.

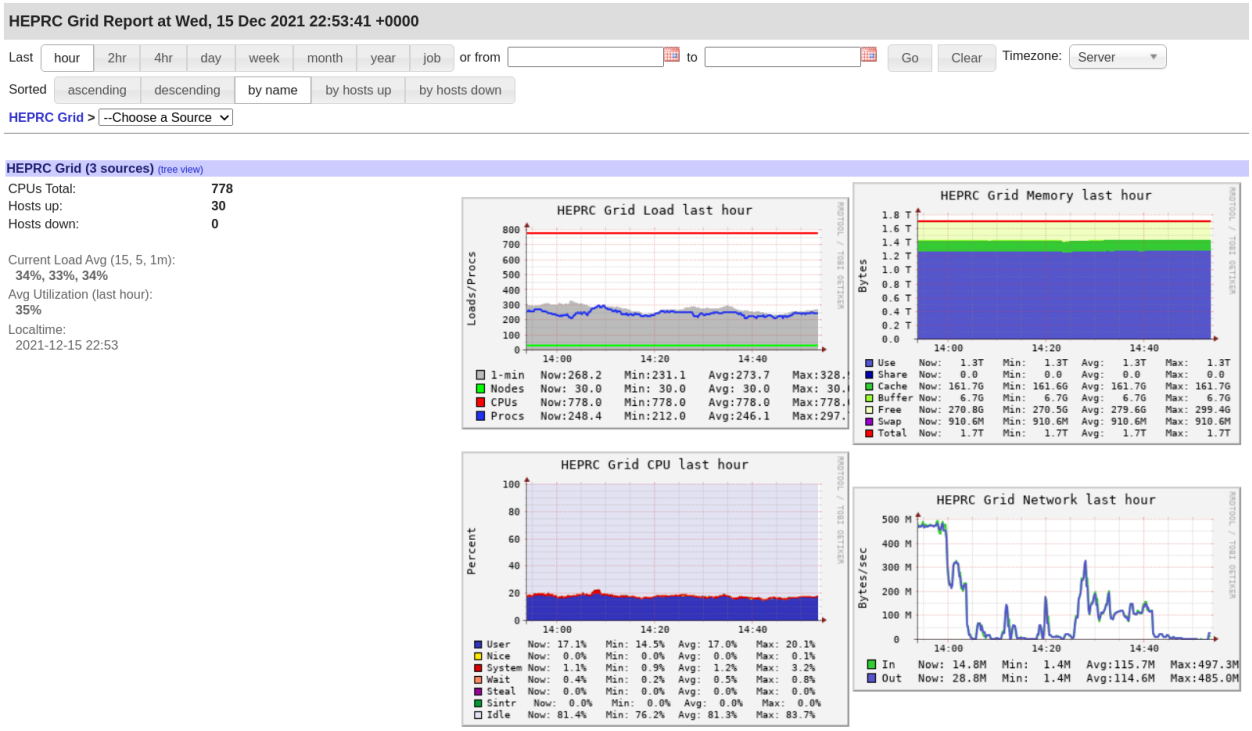

Figure 2: Ganglia web interface showing overview of the HEPRC grid.

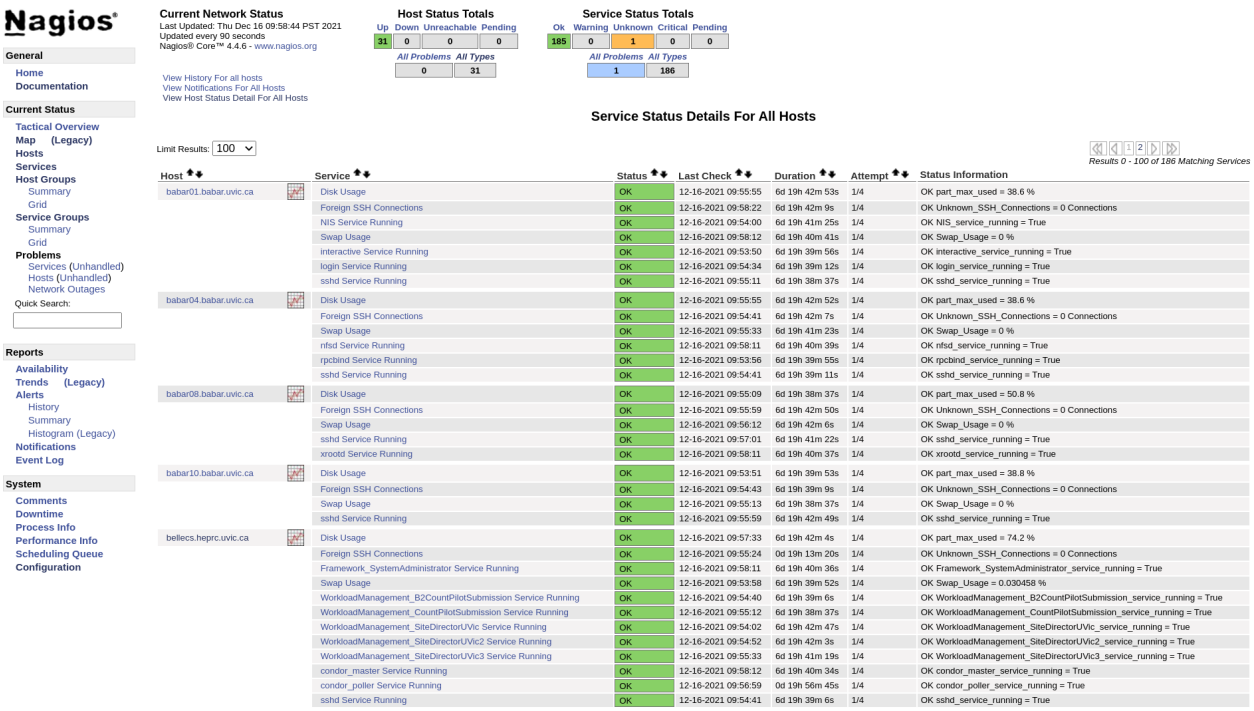

Figure 3: Nagios web interface showing services.

Each logical group of hosts with similar functions or network topology is organized into a *cluster*. One single `gmond` instance in the cluster is designated as the collector `gmond`, whereas the remainder of the machines report metrics to it. Multiple clusters are organized into *grids*. This mitigates the limitation that each cluster can include no more than 2000 monitored hosts [30]. Since each `gmetad` instance is accompanied by a `gweb` component, a central `gmetad` instances that oversees the whole system will poll only summary statistics from the grids, and details can be further investigated by visiting the web page associated with the specific grid [36]. Using the structure shown in Figure 4, the system can be made to scale to a large number of hosts.

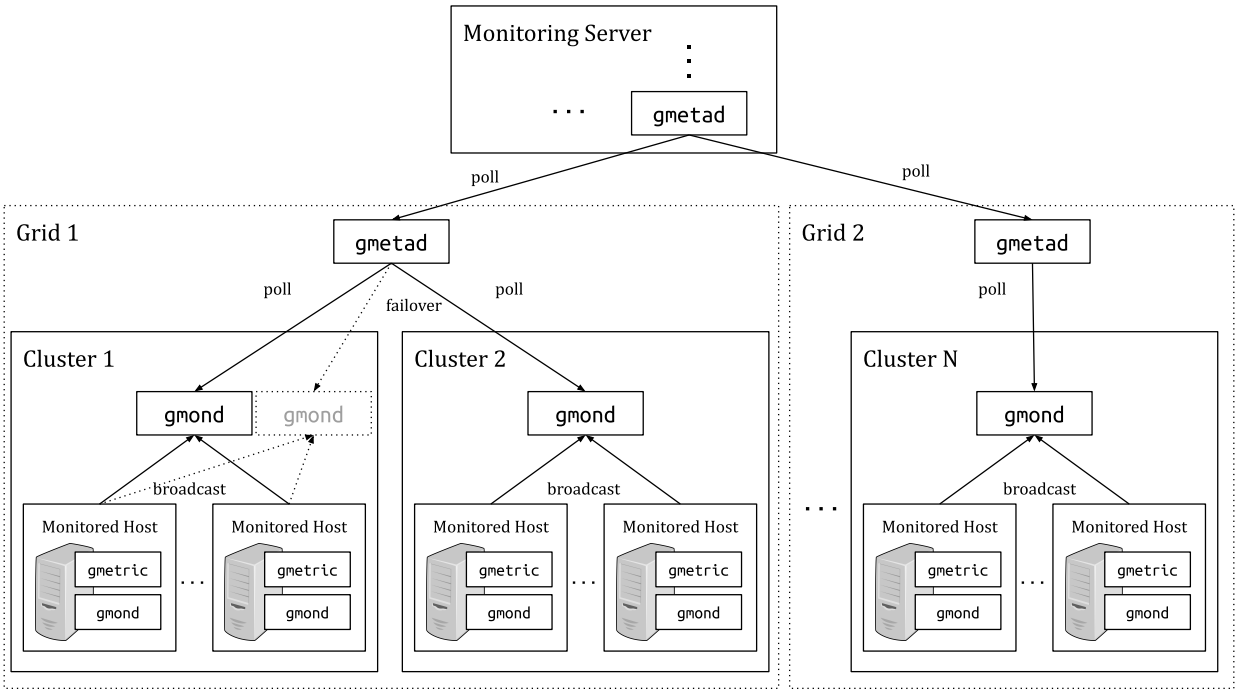

Figure 4: Diagram of the scalability of the monitoring system.

In addition to the metrics gathered by Ganglia by default, the system must collect information on the number of unauthorized users connected via SSH and state information on running processes. To enable this, Ganglia has two separate methods of adding custom metrics. First, custom metrics can be added as Python [37] modules. However, this feature is only compatible with Python 2, which has hit end of life as of January 1, 2020 [38]. Therefore, `gmetric` is used instead. A shell script, listed in Appendix A, is run at regular intervals via cron and invokes `gmetric` to send a custom metrics to the monitoring server. The processes that are monitored are stored in a configuration file.

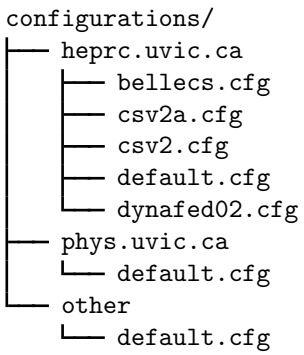

Figure 5: Sample configuration directory structure.

2.4 Deployment

During deployment, several steps must be completed:

1. The `gmond` daemon must be installed and configured on the target machine
2. The custom process monitoring script along with the configuration file containing the processes to monitor must be sent the machine
3. The `gmond` daemon must be started and the custom script must be configured to run at predefined intervals.

To deploy the monitoring system to machines in the HEPRC infrastructure, a script (listed in Appendix B) was developed to perform all steps necessary for remote installation over SSH. Since each monitored machine has different processes that are critical to its operation, a configuration file must first be created that defines these processes. These configurations files are stored in a directory structure similar to the one listed in in Figure 5. The directories represent the domain names of the hosts, while the `.cfg` files represent the specific hosts. If the domain of the new server is known, it will load the appropriate `domain/host.cfg` file in that directory if it exists as well as the `domain/default.cfg`. If the domain name is not known, `other/default.cfg` will be used. During deployment, the script will send this information to the target machine. Finally, Nagios must also be aware of this configuration, thus the configuration directory structure must be synced with the monitoring server.

New servers added to the Nagios configuration are built up from the host configuration files such as those shown in Figure 6 using a set of templates that are listed in Appendix C. The first section defines the processes that need to be monitored, while the second defines alerts based on a metric name, an operator, and

```
[PROCESS]
sshd
condor_poller
condor_master
[COMMAND]
part_max_used!more!90 Disk Usage
Swap_Usage!more!50 Swap Usage
Unknown_SSH_Connections!notequal!0 Foreign SSH Connections
```

Figure 6: Sample host configuration file.

a threshold. This system makes it possible to set the configuration for both the custom Ganglia monitoring script on the host machine and Nagios on the monitoring server in one place.

2.5 Host management

Nagios is only aware of hosts that are explicitly specified in its configuration files and must be restarted for hosts to be added or removed. However, Ganglia is able to add and remove hosts from its monitoring system dynamically. Therefore, a set of tools was developed in order to make the process of syncing hosts between Ganglia and Nagios simple and automated. These tools consist of PHP files which integrate into the Nagios web interface to add and remove hosts, as well as shell scripts that query a custom Ganglia extension, `requestv2`, that was written to provide API-level access to information stored in Ganglia over HTTP. Hence, the hosts that are currently in Ganglia but not in Nagios can be added using the configuration files that were sent from the deployment server.

3 Anomaly Detection

Although Nagios can send an alert when something is going wrong, these simple state-based and threshold-based alerts may not catch all instances of host behaviour that deviate from the norm. Since Nagios only checks one metric at a time, it cannot detect complex anomalies that involve multiple metrics. Additionally, a large amount of data is collected by the monitoring system, and unless an administrator knows specifically what to look at, there is a large possibility that critical faults can go undetected. Formal anomaly detection techniques are therefore the ideal candidate to use to report such occurrences to system administrators.

3.1 Background

Anomaly detection in this case involves creating a decision function that, given a k-dimensional data point where k is the number of metrics, can generate an anomaly score (where a higher value is considered more anomalous). A binary prediction function can then be defined to take a threshold value, and label any scores above the threshold as outliers, and the rest as inliers. The threshold is calculated based on the contamination (proportion of outliers) of the training data [39]. For models that can model the temporal dimension, such as some machine learning models, data will be provided in the form of *Multivariate Temporal Windows*, which are $k \times w$ matrices where k is the number of metrics and w is the number of data points for each metric [40]. For traditional models that are not able to represent time, the window is extended into a $k \cdot w$-dimensional vector, however part of the correlation is lost in this transformation [40].

In the case of monitoring machines in a data center or otherwise, anomalies are deviations in the behaviour of a machine based on previous performance data generated by the machine itself or similar machines in the infrastructure. This can be achieved by applying either traditional algorithms such as IForest [41], or machine learning algorithms such as LSTM Autoencoders [42].

3.2 Anomaly detection in the CERN infrastructure

The CERN infrastructure has a similar problem to the one described above, albeit at a larger scale. In [40], CERN machines are described as either part of the "batch" or "shared" dataset. These distinctions indicate primarily the regularity of load experienced, where "shared" machines have a more sporadic load profile then "batch" machines do. The machines are further organized into host groups, with the assumption that machines that are in the same host group will have similar performance characteristics. Thus, one week of training data is gathered over several performance metrics for all machines in a particular host group. The data is then normalized by coefficients calculated for all of the data per metric in the given host group. Next, a detection model is trained on this data. Finally, the model can now be used to generate predictions.

3.3 Anomaly detection in the HEPRC infrastructure

Since the HEPRC's monitoring system is different then the one used in CERN, the same data preparation pipeline cannot be used. A new system was therefore created as seen in Figure 7.

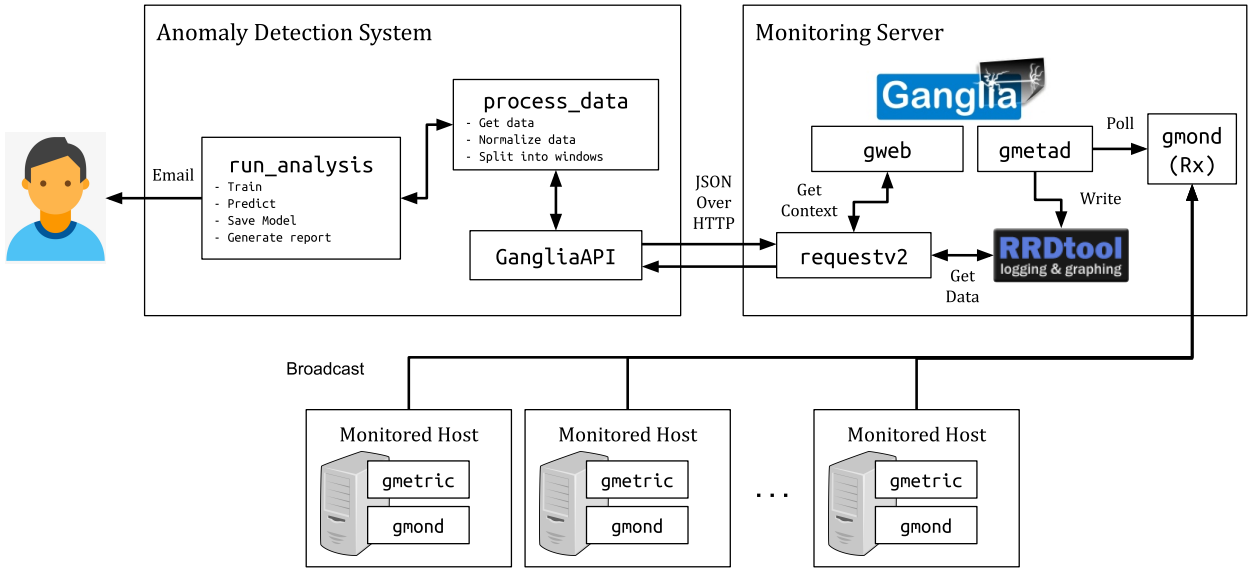

Figure 7: Diagram of the HEPRC anomaly detection system.

3.3.1 Data preparation

Data is gathered directly from the RRD files that store the time series data on the monitoring server using `requestv2`, a custom extension to `gweb` that provides access to the time series data given a cluster, host name, list of metrics, start and end times, as well as the mean aggregation resolution. By default, the RRD files created with Ganglia are mean aggregated in stages: first over 15 seconds, then 1 minute and finally 10 minutes. This makes it convenient to gather data at 10 minute intervals as is done in the original paper without any additional processing required. This way, there are 144 data points generated per metric per day for each host.

Since most of the machines in the HEPRC infrastructure are not explicitly sorted into hostgroups as they are in CERN[2], training data must be gathered per host rather than per host group. The training interval is typically set at one week before the day that the inference will be performed on, however the system supports variable-length training intervals as well as setting a fixed interval where it is clear that the machine was performing as expected. This, along with the set of metrics to gather for each host are defined in the

[2]Other than the cloud worker nodes, which may have more self-similar behaviour.

configuration file listed in Appendix D. The data for the training interval is normalized by metric, then split into overlapping 8 hour (48 data point) long windows, one starting at each data point.

3.3.2 Model training

In [40], the IForest model from the PyOD [43] module scores highest when tested on the "Batch" dataset, whereas the AutoEncoder LSTM model implemented in [42] scored the highest in the "Shared" dataset. Therefore, the system created has the capability to use either of these models, as well as any model that implements the PyOD `BaseDetector` API. The model used is specified in the configuration file. One model per host is then trained on the processed training data and saved.

3.3.3 Inference and reporting

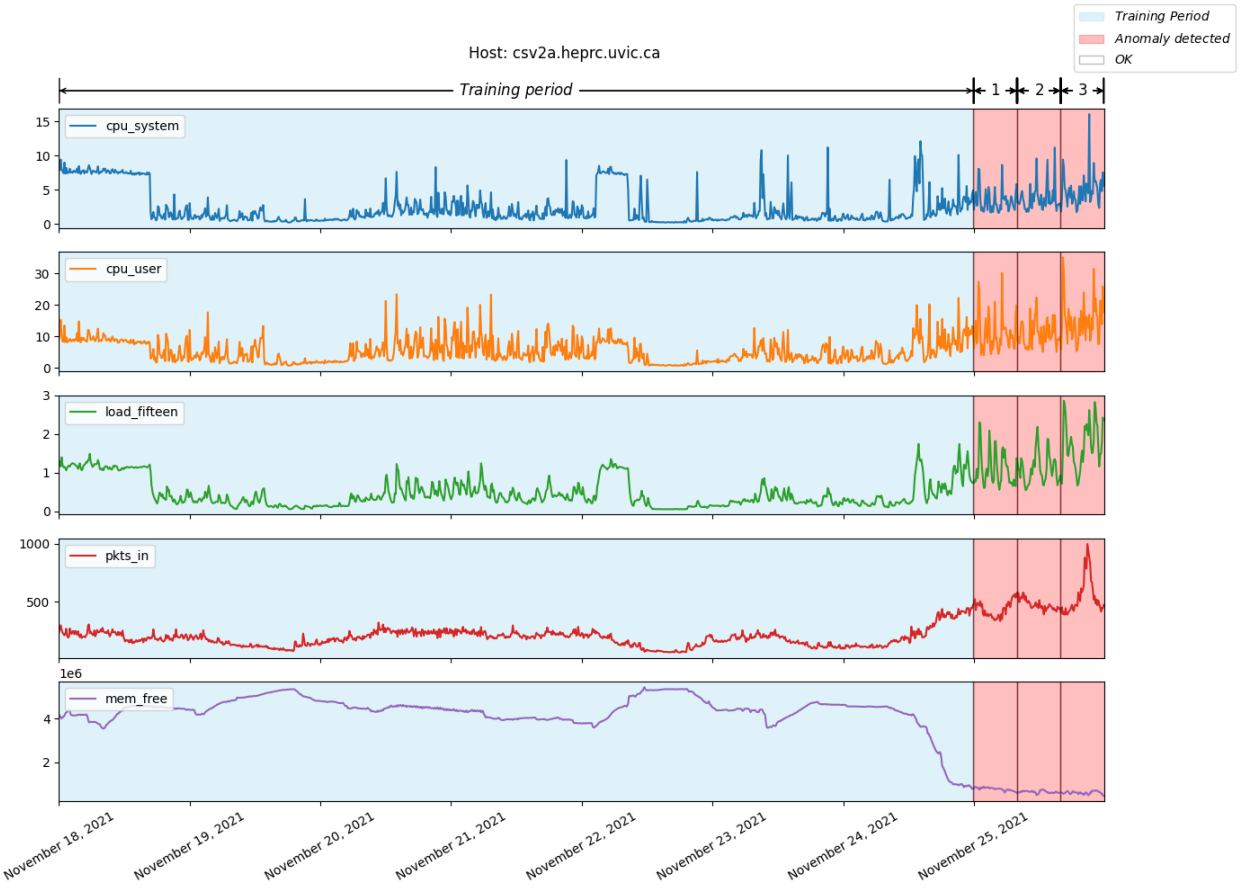

Figure 8: Sample plot of anomalous host.

Finally, data can be gathered for the day that will be predicted on. If day has passed, then all 3 of the non-overlapping 8 hour windows are available for inference. Otherwise, the system will use as many windows as are currently available. Applying the model to the inference data per host creates a report such as the one shown in Appendix E. If any hosts present as anomalous during the inference period, a plot is generated from the training and inference data that indicates in which window the anomaly has occurred as shown in Figure 8. The report and plots are then sent as an email to the primary contact in the configuration. When a system administrator receive this report, they are able to determine at a glance what the issue may be from the graph and take action if needed.

4 Conclusions

After considering many different software solutions for the remote monitoring of the HEPRC infrastructure and creating several test configurations, a system that integrated both Nagios and Ganglia was determined to be the best fit. A set of custom tools were developed to integrate the two together and the system was deployed. Complementary to the alerting provided by Nagios, a system that leverages anomaly detection algorithms and machine learning to automatically detect complex anomalies and alert administrators was created based on the work done at CERN.

5 Future Work

Now that this monitoring system as been established, more hosts can be added, especially more cloud worker nodes. In addition, more custom metrics can be sent to Ganglia so that relevant aspects of software running on the HEPRC infrastructure can be monitored. Additional work can also be done on the anomaly detection system in order to increase the prediction accuracy. One method to accomplish this would be to create a labelled validation dataset that can be used to objectively evaluate different models, choices of metrics, hyperparameters, as well as different strategies for selecting training periods. Since the HEPRC's use case is ultimately different from the one at CERN, separate testing must be done.

Finally, using application-specific metrics such as those provided by HTCondor [44] would give more insight into the state of the applications that are currently running on the host, which may improve the

anomaly detection's system ability to identify outliers.

Acknowledgements

I would like to sincerely thank Marcus Ebert, Colson Driemel and the other members of the HEPRC group for their support and mentorship during this work term.

References

[1] CERN, "The ATLAS Experiment," *ATLAS Experiment*. [Online]. Available: https://atlas.cern/about/ [Accessed Dec. 14, 2021].

[2] DESY, "Belle II," *Belle II*. [Online]. Available: https://www.belle2.org/ [Accessed Dec. 14, 2021].

[3] F. Berghaus, K. Casteels, C. Driemel, M. Ebert, F. F. Galindo, C. Leavett-Brown, D. MacDonell, M. Paterson, R. Seuster, R. J. Sobie, S. Tolkamp, and J. Weldon, "High-throughput cloud computing with the cloudscheduler vm provisioning service," *Computing and Software for Big Science*, vol. 4, no. 1, p. 4, Feb. 2020. [Online]. Available: https://doi.org/10.1007/s41781-020-0036-1

[4] A. Aimar, A. A. Corman, P. Andrade, J. D. Fernandez, B. G. Bear, E. Karavakis, D. M. Kulikowski, and L. Magnoni, "Monit: Monitoring the cern data centres and the wlcg infrastructure," *EPJ Web Conf.*, vol. 214, p. 08031, Sept. 2019. [Online]. Available: https://doi.org/10.1051/epjconf/201921408031

[5] The Cacti Group, Inc., "About Cacti," *Cacti*. [Online]. Available: https://www.cacti.net/ [Accessed Dec. 10, 2021].

[6] T. Oetiker, "About RRDtool," *RRDtool logging & graphing*. [Online]. Available: https://oss.oetiker.ch/rrdtool/ [Accessed Dec. 10, 2021].

[7] MariaDB Foundation, "MariaDB Server: The open source relational database," *MariaDB*. [Online]. Available: https://mariadb.org/ [Accessed Dec. 10, 2021].

[8] Apache, "Apache HTTP Server Project," *Apache HTTP Server Project*. [Online]. Available: https://httpd.apache.org/ [Accessed Dec. 13, 2021].

[9] Oracle Corporation, "MySQL," *MySQL*. [Online]. Available: https://www.mysql.com/ [Accessed Dec. 13, 2021].

[10] The PHP group, "PHP," *PHP*. [Online]. Available: https://www.php.net/ [Accessed Dec. 13, 2021].

[11] The Cacti Group, Inc., "What is Cacti?" *Cacti*. [Online]. Available: https://www.cacti.net/info/cacti [Accessed Dec. 10, 2021].

[12] The Cacti Group, Inc., "Spine Information," *Cacti*. [Online]. Available: https://www.cacti.net/info/spine [Accessed Dec. 10, 2021].

[13] Cacti, *Thold GitHub repository*. [Online]. Available: https://github.com/Cacti/plugin_thold [Accessed Dec. 10, 2021].

[14] The Munin Project, "What is Munin?" *Munin*. [Online]. Available: https://munin-monitoring.org/ [Accessed Dec. 10, 2021].

[15] The Perl Foundation, "Perl," *Perl*. [Online]. Available: https://www.perl.org/ [Accessed Dec. 13, 2021].

[16] The Munin Project, "Munin's Architecture," *Munin*. [Online]. Available: http://guide.munin-monitoring.org/en/latest/architecture/index.html [Accessed Dec. 11, 2021].

[17] Prometheus, "From metrics to insight," *Prometheus*. [Online]. Available: https://prometheus.io/ [Accessed Dec. 10, 2021].

[18] Prometheus, "What is Prometheus?" *Prometheus*. [Online]. Available: https://prometheus.io/docs/introduction/overview/ [Accessed Dec. 10, 2021].

[19] Google, "Go," *Go*. [Online]. Available: https://go.dev/ [Accessed Dec. 13, 2021].

[20] Prometheus, "Exporters and integrations," *Prometheus*. [Online]. Available: https://prometheus.io/docs/instrumenting/exporters/ [Accessed Dec. 13, 2021].

[21] Grafana Labs, "Grafana," *Grafana*. [Online]. Available: https://grafana.com/grafana/ [Accessed Dec. 10, 2021].

[22] Prometheus, "Alertmanager," *Prometheus*. [Online]. Available: https://prometheus.io/docs/alerting/latest/alertmanager/ [Accessed Dec. 13, 2021].

[23] Elastic NV, "What is the ELK Stack?" *Elastic*. [Online]. Available: https://www.elastic.co/what-is/elk-stack [Accessed Dec. 13, 2021].

[24] Amazon, "The ELK stack," *Amazon Web Services*. [Online]. Available: https://aws.amazon.com/opensearch-service/the-elk-stack/ [Accessed Dec. 13, 2021].

[25] Elastic NV, "Metricbeat overview," *Elastic*. [Online]. Available: https://www.elastic.co/guide/en/beats/metricbeat/current/metricbeat-overview.html [Accessed Dec. 13, 2021].

[26] Elastic NV, "Alerting," *Elastic*. [Online]. Available: https://www.elastic.co/what-is/kibana-alerting [Accessed Dec. 13, 2021].

[27] Zabbix LLC., "Zabbix," *Zabbix*. [Online]. Available: https://www.zabbix.com/ [Accessed Dec. 12, 2021].

[28] Zabbix LLC., "What is Zabbix," *Zabbix*. [Online]. Available: https://www.zabbix.com/documentation/current/en/manual/introduction/about [Accessed Dec. 13, 2021].

[29] Zabbix LLC., "Zabbix Agent," *Zabbix*. [Online]. Available: https://www.zabbix.com/zabbix_agent [Accessed Dec. 13, 2021].

[30] What is Ganglia, "Ganglia Monitoring System," *Ganglia Monitoring System*. [Online]. Available: http://ganglia.sourceforge.net/ [Accessed Dec. 13, 2021].

[31] Nagios Enterprises, "Nagios Core," *Nagios*. [Online]. Available: https://www.nagios.org/projects/nagios-core/ [Accessed Dec. 13, 2021].

[32] Nagios Enterprises, "Active Checks," *Nagios*. [Online]. Available: https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/3/en/activechecks.html [Accessed Dec. 13, 2021].

[33] Nagios Enterprises, "Passive Checks," *Nagios*. [Online]. Available: https://assets.nagios.com/downloads/nagioscore/docs/nagioscore/3/en/passivechecks.html [Accessed Dec. 13, 2021].

[34] J. Linge, "PNP4Nagios Documentation," *PNP4Nagios*. [Online]. Available: http://docs.pnp4nagios.org/start [Accessed Dec. 13, 2021].

[35] The OpenStack Project, "OpenStack," *OpenStack*. [Online]. Available: https://www.openstack.org/ [Accessed Dec. 10, 2021].

[36] M. Massie, B. Li, B. Nicholes, V. Vuksan, R. Alexander, J. Buchbinder, F. Costa, A. Dean, D. Josephsen, P. Phaal, and D. Pocock, *Monitoring with Ganglia: Tracking Dynamic Host and Application Metrics at Scale*. O'Reilly Media, Nov. 2012.

[37] Python Software Foundation, "Python," *Python*. [Online]. Available: https://www.python.org/ [Accessed Dec. 13, 2021].

[38] Python Software Foundation, "Sunsetting Python 2," *Python*. [Online]. Available: https://www.python.org/doc/sunset-python-2/ [Accessed Dec. 10, 2021].

[39] Y. Zhao, "All Models," *PyOD Documentation*. [Online]. Available: https://pyod.readthedocs.io/en/latest/pyod.models.html [Accessed Dec. 13, 2021].

[40] D. Giordano, M. Paltenghi, S. Metaj, and A. Dvorak, "Anomaly detection in the cern cloud infrastructure," *EPJ Web Conf.*, vol. 251, p. 02011, 2021. [Online]. Available: https://doi.org/10.1051/epjconf/202125102011

[41] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation-based anomaly detection," *ACM Trans. Knowl. Discov. Data*, vol. 6, no. 1, Mar. 2012. [Online]. Available: https://doi.org/10.1145/2133360.2133363

[42] M. Paltenghi, "Time Series Anomaly Detection for CERN Large-Scale Computing Infrastructure," M. S. thesis, Politecnico di Milano, Milan, Italy, 2020. [Online]. Available: https://cds.cern.ch/record/2752641

[43] Y. Zhao, Z. Nasrullah, and Z. Li, "Pyod: A python toolbox for scalable outlier detection," *Journal of Machine Learning Research*, vol. 20, no. 96, pp. 1–7, 2019. [Online]. Available: http://jmlr.org/papers/v20/19-011.html

[44] University of Wisconsin–Madison, "Computing with HTCSS," *HTCondor Software Suite*. [Online]. Available: https://research.cs.wisc.edu/htcondor/ [Accessed Dec. 12, 2021].

Appendix A Custom Metric Script

Figure 9 lists the script to send custom metrics from the monitored hosts using `gmetric`.

```
#!/usr/bin/env bash

METRIC_TTL=905
PROC_CFG=/etc/ganglia/host.cfg

ssh_whitelist=( 'uvic.ca' )

# Commands to gather metrics

proc_verify() {
        if [ $(ps axf|grep -c "$1") -gt 1 ];then echo "True"
        else                                     echo "False" ; fi
}


check_ssh_connections() {
  unknown=0
  for host in $(last -w | grep "still logged in" | awk '{print $3}' | uniq);
  do
      hostname=$(getent hosts $host | awk '{print tolower($2)}')
      [[ $hostname =~ ^.*($ssh_whitelist)$ ]] || unknown=$((unknown+1))
  done
  echo $unknown
}

SWAP_PERCENT=$(free -t | grep -i swap|awk '{if ($2 + 0 != 0) {print $3/$2*100} else {print 0}}')

# TYPES: string|int8|uint8|int16|uint16|int32|uint32|float|double

gmetric -n 'Swap_Usage' -v $SWAP_PERCENT -t 'uint16' -u '%' -d $METRIC_TTL
gmetric -n 'Unknown_SSH_Connections' -v $(check_ssh_connections) -t 'uint16' -u 'Connections' -d
↪  $METRIC_TTL

while read proc ; do
  if [ $proc != "" ];
  then
    gmetric -n $proc'_service_running' -v $(proc_verify $proc) -t 'string' -u '' -d $METRIC_TTL
  fi
done < $PROC_CFG
```

Figure 9: Custom metric script.

Appendix B Deployment Script

Figure 10 lists the script used to deploy Ganglia monitoring to new hosts.

```
#!/usr/bin/env bash

host=$1
domain=$2
port=$3
user=$4

if [ "$#" -ne 4 ] ; then echo "USAGE: deploy_ganglia.sh hostname domain port user" && exit 1 ; fi

# Send configuration files/scripts to remote host

scp -P$port gmond.conf monitor.sh $user@$host.$domain:/tmp

# Process config file

awk '/\[PROCESS\]/{flag=1;next}/\[COMMAND\]/{flag=0}flag' configurations/$domain/$host.cfg | ssh
↪  -p$port $user@$host.$domain "cat > /tmp/$host.cfg"

# Run installation

ssh -tt -p$port $user@$host.$domain "sudo yum -y install epel-release; sudo yum -y install
↪  ganglia-gmond ganglia-gmond-python; sudo mv /tmp/gmond.conf /tmp/monitor.sh /tmp/host.cfg
↪  /etc/ganglia; sudo chmod +x /etc/ganglia/monitor.sh; echo \"echo '*/5 * * * * root /bin/sh
↪  /etc/ganglia/monitor.sh' > /etc/cron.d/ganglia\"|sudo bash; sudo service gmond restart gmond;
↪  sudo systemctl enable gmond; sudo /etc/ganglia/monitor.sh &>/dev/null;"

# Send config file to nagios server

r_ip=?   # Replace "?" with ip of nagios server
r_user=? # Replace "?" with username on nagios server
r_port=22

rsync -crpz -e "ssh -p$r_port" configurations $r_user@$r_ip:/etc/nagios

exit 0
```

Figure 10: Deployment script.

Appendix C Sample Nagios Templates

Figures 11, 12, and 13 list the templates used to generate new Nagios object configuration files.

```
define host {
        use             ganglia-linux-host
        host_name       {{HOSTNAME}}
        address         {{HOSTADDR}}
}
```

Figure 11: Generic host template.

```
define service {
        use             ganglia-service
        name            {{HOSTNAME}}-{{METRIC}}
        description     {{DESCRIPTION}}
        check_command   check_ganglia_metric!{{ARGSTR}}
        host_name       {{HOSTNAME}}
}
```

Figure 12: Generic service template.

```
define service {
        use             ganglia-service
        name            {{SERVICE}}-{{HOSTNAME}}
        description     {{SERVICE}} Service Running
        check_command   check_ganglia_metric!{{SERVICE}}_service_running!notequal!'True'
        host_name       {{HOSTNAME}}
}
```

Figure 13: Generic process template.

Appendix D Sample Anomaly Detection Configuration

Figure 14 lists a sample configuration file for the anomaly detection system.

```
[config]
monitor_server  http://206.12.154.221/ganglia/
detector        pyod.models.iforest.IForest

admin_email     vkamel@uvic.ca
max_workers     2   # Number of parallel processes to start
train_days      7   # Number of days to train on

[model]             # Model hyperparameters (IForest)
n_estimators    300
contamination   0.05
verbose         1
max_samples     'auto'
max_features    1.0
random_state    3393

[manifest]
# Cluster       Hostname                Metrics
infrastructure  dynafed02.heprc.uvic.ca cpu_system,cpu_user,load_fifteen,pkts_in,mem_free
infrastructure  csv2a.heprc.uvic.ca     cpu_system,cpu_user,load_fifteen,pkts_in,mem_free
infrastructure  csv2-dev2.heprc.uvic.ca cpu_system,cpu_user,load_fifteen,pkts_in,mem_free
infrastructure  bellecs.heprc.uvic.ca   cpu_system,cpu_user,load_fifteen,pkts_in,mem_free
infrastructure  ganglia.heprc.uvic.ca   cpu_system,cpu_user,load_fifteen,pkts_in,mem_free
infrastructure  shoal.heprc.uvic.ca     cpu_system,cpu_user,load_fifteen,pkts_in,mem_free
```

Figure 14: Anomaly detection configuration.

Appendix E Sample Anomaly Detection Report

Figure 15 lists a sample report generated by the anomaly detection system.

```
NOTICE

Anomalies have been detected in the following hosts:
  - csv2a.heprc.uvic.ca
    (http://206.12.154.221/ganglia/?c=infrastructure&h=csv2a.heprc.uvic.ca&r=week)
  - csv2-dev2.heprc.uvic.ca
    (http://206.12.154.221/ganglia/?c=infrastructure&h=csv2-dev2.heprc.uvic.ca&r=week)
  - ganglia.heprc.uvic.ca
    (http://206.12.154.221/ganglia/?c=infrastructure&h=ganglia.heprc.uvic.ca&r=week)

URL of Ganglia monitoring server:  http://206.12.154.221/ganglia/
-----
Report for day: 2021-11-25
Generated: 2021-12-07 07:53:34
Generated by: heplw67.phys.uvic.ca

Windows:
1 = 00:00:00 to 08:00:00, 2 = 08:00:00 to 16:00:00, 3 = 16:00:00 to 24:00:00

HOST: dynafed02.heprc.uvic.ca
  Window 1:  P(Outlier) =  24.0 %  Thresh = 0.0  Score = -0.23967  Verdict: OK
  Window 2:  P(Outlier) =  55.1 %  Thresh = 0.0  Score = -0.12573  Verdict: OK
  Window 3:  P(Outlier) =  50.6 %  Thresh = 0.0  Score = -0.14218  Verdict: OK
HOST: csv2a.heprc.uvic.ca
  Window 1:  P(Outlier) = 100.0 %  Thresh = 0.0  Score =  0.04880  Verdict: Anomaly
  Window 2:  P(Outlier) = 100.0 %  Thresh = 0.0  Score =  0.05061  Verdict: Anomaly
  Window 3:  P(Outlier) = 100.0 %  Thresh = 0.0  Score =  0.11097  Verdict: Anomaly
HOST: csv2-dev2.heprc.uvic.ca
  Window 1:  P(Outlier) =  90.8 %  Thresh = 0.0  Score = -0.00249  Verdict: OK
  Window 2:  P(Outlier) = 100.0 %  Thresh = 0.0  Score =  0.11603  Verdict: Anomaly
  Window 3:  P(Outlier) = 100.0 %  Thresh = 0.0  Score =  0.05279  Verdict: Anomaly
HOST: bellecs.heprc.uvic.ca
  Window 1:  P(Outlier) =  35.5 %  Thresh = 0.0  Score = -0.10987  Verdict: OK
  Window 2:  P(Outlier) =  75.7 %  Thresh = 0.0  Score = -0.01728  Verdict: OK
  Window 3:  P(Outlier) =  60.8 %  Thresh = 0.0  Score = -0.05162  Verdict: OK
HOST: ganglia.heprc.uvic.ca
  Window 1:  P(Outlier) =  52.8 %  Thresh = 0.0  Score = -0.06453  Verdict: OK
  Window 2:  P(Outlier) =  98.5 %  Thresh = 0.0  Score =  0.00453  Verdict: Anomaly
  Window 3:  P(Outlier) =  58.9 %  Thresh = 0.0  Score = -0.05539  Verdict: OK
HOST: shoal.heprc.uvic.ca
  Window 1:  P(Outlier) =  59.8 %  Thresh = 0.0  Score = -0.05895  Verdict: OK
  Window 2:  P(Outlier) =  68.0 %  Thresh = 0.0  Score = -0.04424  Verdict: OK
  Window 3:  P(Outlier) =  82.7 %  Thresh = 0.0  Score = -0.01771  Verdict: OK
```

Figure 15: Anomaly detection report.