University of Victoria
Faculty of Engineering
Spring 2016 Work Term Report

Dynamic Monitoring and Alerting for High Throughput Computing Clouds

Department of Physics
University of Victoria
Victoria, BC

Darryl Ring
V00182679
Work Term 3
Electrical Engineering
ringda@uvic.ca

May 2, 2016

In partial fulfillment of the requirements of the
Bachelor of Engineering Degree

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

I approve the release of this report to the University of Victoria for evaluation purposes only.
The report is to be considered (**select one**): ☐ NOT CONFIDENTIAL ☐ CONFIDENTIAL

Signature: _____ Position: _____ Date: _____

Name (print): _____ E-Mail: _____ Fax #: _____

If a report is deemed CONFIDENTIAL, a non-disclosure form signed by an evaluator will be faxed to the employer. The report will be destroyed following evaluation. If the report is NOT CONFIDENTIAL, it will be returned to the student following evaluation.

Contents

List of Figures

Dynamic Monitoring and Alerting for High Throughput Computing Clouds

Darryl Ring
ringda@uvic.ca

May 2, 2016

Abstract

The High Energy Physics Research Computing group utilizes both dedicated and commercial cloud computing platforms to provide data processing for high energy physics experiments. Unlike traditional computing infrastructure, cloud computing is dynamic and reconfigurable. Hosts that were once individual physical computers are now virtual machines that can be created or destroyed and started or stopped frequently to meet demand. Monitoring such a dynamic infrastructure requires tools that are equally dynamic.

Many standard monitoring and alerting systems were designed to meet the needs of traditional computing where all hosts and services to be monitored are predetermined. While these systems can meet some of the needs of cloud computing, they cannot readily adapt to a changing environment. Instead, we look to new platforms such as Elasticsearch and Sensu, and new ways to make use of existing systems such as Ganglia and Graphite. In addition, custom components and extensions that enhance these systems for these specific applications are described.

1 Report Specification

The report provides an overview of a monitoring and alerting infrastructure for clouds running high energy physics (HEP) computing jobs. An overview of the software packages used as well as a discussion of custom components developed is provided. Finally, the limitations of this infrastructure, problems encountered during its development, and a direction for future work are discussed.

1.1 Audience

This report is intended for members of the High Energy Physics Research Computing group at the University of Victoria, and future co-op students.

1.2 Prerequisites

A general understanding of cloud computing, virtualization, and web technologies is assumed.

2 Introduction

The High Energy Physics Research Computing (HEPRC) group at the University of Victoria makes use of cloud computing to support particle physics experiments, such as the ATLAS experiment at the Large Hadron Collider at CERN and Belle-II detector at KEK in Japan [1]. These experiments produce large data sets that require processing, and cloud computing allows this processing to occur whenever and wherever resources are available. The experiments are therefore able to make use of public, private, commercial, and research cloud computing resources, and to ramp this usage up and down with fluctuations in workload.

Making the most of this dynamic infrastructure requires the proper operation of many distinct but interdependent components. If one component is not operating correctly, the system as a whole may not perform adequately, or may not perform at all. Monitoring these systems must therefore be an integral component of any cloud computing infrastructure, and the monitoring system itself must be dynamic and reliable.

2.1 Monitoring

In this context, the term *monitoring* encompasses three main functions: collecting and analyzing system metrics, evaluating the status and health of systems and services, and alerting administrators to situations that may require manual intervention. Additional functions related to monitoring may include the automated correction of abnormal conditions, centralization and analysis of log data, and visualization of various performance metrics. These functions are discussed in more detail in section 3, and some of the monitoring tools that were evaluated are introduced.

2.2 Problem Definition

The HEPRC group maintains several separate *instances* for different experiments. Each instance consists of a single master server running the Cloud Scheduler cloud management [2] and HTCondor workload management software [3]. Compute jobs are submitted to HTCondor, and Cloud Scheduler starts and stops VMs as necessary across multiple clouds to meet the workload requirements. Each instance may make use of the same clouds, but the resources are managed separately.

The existing monitoring infrastructure was primarily based around a customized version of Munin [4], named Munin Collector, which collects system and performance metrics [5]. In addition, a custom script generated a textual visualization of the current status of each instance. A more complete and cohesive monitoring system was desired which incorporated the features introduced in the previous section. In addition to these basic functions, however, a monitoring system for this infrastructure would also need to meet additional requirements.

2.3 Requirements

Before the process of evaluation and development was initiated, a set of requirements for a monitoring system was presented. These requirements specified that the monitoring system must:

1. be open source and self-hosted;
2. support the dynamic addition and removal of monitored hosts and services;
3. be able to gather information from VMs on private IPs;
4. provide real-time charting and data;
5. provide alerting and triggering;
6. allow for extensibility in the form of custom plug-ins; *and,*
7. support automated deployment and configuration;

Taken together, requirements 2 and 3 imply that monitoring data must be "pushed" from the hosts to the monitoring server. If a VM is not publicly accessible (i.e. it is behind a firewall which does not permit inbound connections, or it does not have a public IP), a monitoring server cannot send commands or requests for data to it. It is therefore desirable that monitored hosts schedule their own monitoring tasks and send data when it is available. The monitoring server must be able to accept data from an unknown host and process it accordingly. Requirements 2 and 3 proved to eliminate many monitoring tools from consideration. Requirement 7 means that the installation and configuration of the monitoring systems should not require human intervention. This implies that the configuration should be file or API-based, and not require a user to navigate a graphical interface.

In addition to these mandatory requirements, there was a stated preference for solutions that meet certain criteria. These were that the selected components be reasonably well established with a large install base

and active community. In open source software, this is an indicator of stable software with good support. It was also desired that the system be relatively simple overall, and easy to install, configure, and maintain. To be useful, a monitoring system should be more stable than the systems it must monitor.

Using these requirements, many monitoring tools were evaluated. In some cases, this evaluation was cursory as it was determined early that they would not meet these requirements. In other cases, the evaluation involved installing, configuring, and running the tools across one or more hosts. This evaluation process is discussed further in section 3.7.

3 Monitoring Tools

As was introduced in section 2.1, a monitoring system should provide three primary functions: collection and aggregation of metrics, status and health checks, and alerting. In addition, automated remediation, log centralization, and visualization are desired. A monitoring system must therefore integrate the components necessary to provide these functions.

In the sections that follow, these functions will be described in greater detail, and some of the available monitoring tools that were evaluated are introduced. Section 3.7 discusses the approach taken to the evaluation of these tools, and lists the tools selected.

3.1 Metrics

There are many metrics that can provide insight into the operation of a computer system. These metrics can measure the performance of a system, indicate resource over- or under-utilization, and help administrators locate bottlenecks and improve overall efficiency. Metrics can locate issues with specific hosts or, when aggregated across many hosts, can identify issues with a system as a whole. Without adequate metrics, it is difficult to make informed decisions about how a system is functioning or how to improve it.

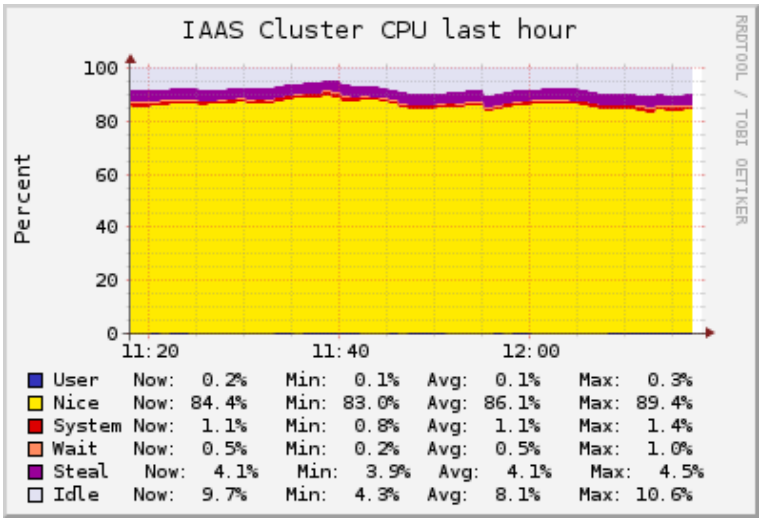


Figure 1: CPU metrics aggregated across multiple hosts, generated by Ganglia

Ideally the collection of metrics should simply involve the reading of values from the host. For example, the current load average, number of running processes, CPU utilization, or free system memory are all easily available, and can be read directly. Other metrics, such as the ping time to a remote host or the number of rows in a database table, require some processing in order to determine, but these should have minimal impact on the system. Measuring the system should not interfere with its behaviour.

Metric collection and aggregation is typically implemented using a client-server model. A client runs on the host, collects metrics, and transmits them to the server. The server aggregates and stores the metrics and makes them available for visualization and analysis. The client should be lightweight so that it meets the requirement of minimal impact. A cloud computing environment requires a system that can adapt to change: what metrics to collect and how to collect them must be predefined on the server or client, but the hosts submitting metrics must be able to come and go.

Many tools exist for metric collection, since it is likely the most widely implemented monitoring function. **Munin** [4] was currently in use across HEPRC servers and VMs. **Ganglia** [7] was also installed, but not in use by the HEPRC group. These are the most common monitoring tools for gathering metrics, and provided a baseline for evaluating other tools such as **Telegraf** [10] and **Topbeat** [14].

3.1.1 Munin

Munin, active since 2003, is a monitoring system that collects metrics from remote servers and aggregates them for display on a master server [4]. It does not support "pushing" of data from the remote servers, and thus cannot be readily used on VMs that are not publicly accessible. HEPRC is currently using a custom version of Munin which has this capability (among other changes) [5]. Ultimately, Munin does not adequately meet the requirements for this project. Instead of maintaining a separate custom version of Munin, an alternate monitoring tool was desired.

3.1.2 Ganglia

Ganglia, first released in 2000, is a distributed monitoring tool used to collect system metrics and aggregate them across many hosts [7]. While the original application of Ganglia was grid computing, its distributed design allows it to adapt to the dynamic nature of cloud computing. Monitored hosts are grouped into *clusters* and clusters can be grouped into *grids*. Metrics can thus be aggregated across clusters and grids to provide an overall picture of a compute cloud's performance.

Ganglia consists of three components: the monitoring daemon, `gmond`; the meta daemon, `gmetad`; and the web front-end, `ganglia-webfrontend`. `gmond` collects system metrics and transfers them over the network. It supports unicast (sending data to a single location) or multicast (sending and receiving data from multiple locations) transmission which allows for flexible, distributed monitoring [8]. Each `gmond` can both collect system metrics from its host as well as from `gmond` instances on other hosts. `gmond` does not, however, aggregate or store metric data.

Instead, `gmetad` collects metrics from `gmond`s and other `gmetad`s, aggregates them, and stores them. A `gmetad` is thus the endpoint for all metrics generated by `gmond`s. Finally, `ganglia-webfrontend` communicates with `gmetad`s to generate charts and provide other visualizations. These visualizations can display data from individual hosts, or aggregated across clusters, grids, or any arbitrary grouping.

Ganglia was found to be easy to install as many Linux distributions include Ganglia in their default package repositories. CERN has deployed Ganglia in production, and includes Ganglia in CernVM (the VM image in use by the experiments) by default [9]. This made Ganglia an ideal candidate, as it provided the desired functionality and met the requirements of this project.

3.1.3 Telegraf

Telegraf is a component in the InfluxDB stack, which aims to provide a complete monitoring and analytics system [10]. It is very new, having been first released in 2015 [11]. Aside from being very new and not being included in CernVM or default package repositories, Telegraf and InfluxDB meet all of the requirements for this project. Telegraf runs as a lightweight service on each host to be monitored, and is configured with the hostname and port of the InfluxDB server to which data should be sent.

The InfluxDB stack also includes Chronograf, for "time-series data visualization" [12], and Kapacitor, for "time-series data processing, alerting and anomaly detection" [13]. This stack is compatible with other tools such as Grafana [34], Graphite [28], and Bosun [26], all described in the coming sections. These tools

are also very new, which is not ideal as their APIs are in flux. While stable 1.0 releases are anticipated this year, implementing a monitoring system using software that is possibly unstable and very likely to change would be counterproductive.

3.1.4 Topbeat

Topbeat [14] is a component of the Elasticsearch [37] stack described in section 3.6.1. It collects a number of system metrics and pushes them to Elasticsearch for storage. Topbeat itself is easy to install and configure, though it is not normally included in CernVM, or available in default package repositories, and thus requires extra steps to install. Elasticsearch, discussed further in section 3.6.1, is not ideal for the storage of large amounts of time-series numerical data, however. It is instead optimized for indexing text data and full-text search. Visualizing and analyzing these data across many hosts was found to be challenging without a large amount of custom development.

3.2 Status and Health

The status or health of a system is determined using *checks* which are boolean operations performed on some aspect of the system's state. These can be as simple as answering a yes or no question (e.g. is a specific process running, is a network service accessible) or can involve comparing specific metrics to set values (e.g. is there enough free memory or disk space on a computer). While a single failing check does not necessarily imply a system failure, it may indicate an abnormal state that should be addressed.

One must determine and configure how to run a check, which hosts to run it on, how frequently to run it, and what should happen if it fails. A system which allows flexibility in these check definitions is therefore desirable. In cloud computing, hosts may appear or disappear frequently, so a system requiring a single static configuration will not work. In the ideal case, hosts could appear, determine which checks they must run, publish their check results while operational, and disappear gracefully.

3.2.1 Nagios

Nagios is described as "the industry standard in IT infrastructure monitoring" [15]. This description was found to be accurate during the initial search for monitoring tools to evaluate: as in [16], Nagios is commonly the first suggestion. As such, it was the first monitoring system to be evaluated.

During the evaluation, Nagios was found to be flexible but static. Nagios must know about every host to be monitored at start up, which limits its utility in a cloud computing environment. If check results are published by unknown hosts, they are simply ignored. A common approach is to use some other tool to generate the configuration files from a database of hosts, or use auto-discovery. Given that VMs are starting and stopping minute-by-minute, this approach is unrealistic and would involve considerable overhead.

The popularity of Nagios has led to a number of derivatives, extensions, and systems described as compatible replacements. While each claims to provide substantial improvements over Nagios's functionality and configuration, they were found to carry the same upsides and downsides as Nagios itself: configuration is static, hosts must be predefined, and pushing data requires additional plug-ins and configuration.

It was determined early on that, despite Nagios's popularity and omnipresence, it would not meet the needs of monitoring HEPRC's cloud resources. Since its derivatives primarily follow the same basic patterns, they would most likely not meet the requirements either. As such, the evaluation of monitor tools moved in a different direction.

3.2.2 Zabbix

Zabbix aims to provide a complete, integrated monitoring system, and describes itself as an "all in one solution" [17]. It aims to provide many of the features of Ganglia, Nagios, and other tools in a single package. This makes it easy to install and configure, but has the downside of limiting easy integration with other components. Zabbix is not a monitoring framework, but a complete solution, and it is not possible to replace one part of Zabbix with a different system.

Configuring Zabbix is accomplished through a web interface, which makes use of a relational database back-end. There is also an API that can be used, though it is largely intended for use by extensions. Zabbix supports "active agent auto-registration" which means that any host running a Zabbix agent can register itself with the Zabbix server automatically [18]. While these attributes meet the requirements of this project, Zabbix was found to be cumbersome and inflexible.

3.2.3 Sensu

Sensu is a relatively young alternative to Nagios which has been an active project since July 2011 [20]. It provides similar functionality to Nagios but allows for a much more dynamic and flexible configuration. Sensu can execute checks, collect metrics, and send alerts and notifications. It does not require that clients be predefined, which allows Sensu to meet the requirements of this project. Sensu requires that a client be installed on all hosts to be monitored, which is provided along with the server in a single package. This package is not available from default package repositories, and thus requires extra steps to install.

The Sensu server is built upon Redis [21], an in-memory key-value data store, and the RabbitMQ message broker [22]. While this is more complex than running a single service, these packages are stable, well-supported, and included in default package repositories. Clients communicate with the server over RabbitMQ message queues, which allows both the server and the clients themselves to be simpler. Sensu is Ruby-based, which requires a Ruby interpreter, and the Sensu package includes an embedded interpreter [23].

Sensu runs checks that are simply commands executed in a shell configured using JavaScript Object Notation (JSON). They can thus be written in any programming language. Since Cloud Scheduler is written in Python, and HTCondor includes a Python component, this is convenient. Failing checks create events which are sent to handlers. Handlers are defined in the same way as checks, and can also execute any command in a shell. This is also convenient, since a handler could interact directly with an API that is written in a specific language.

3.2.4 Riemann

Riemann is a completely different approach to monitoring than Nagios and Sensu. It "aggregates events from your servers and applications with a powerful stream processing language" [25]. It can accept data from many sources, provides powerful tools for analyzing them, and can then output them to various destinations or generate alerts. Time-series data points are treated as streams of events that can be filtered, combined, aggregated, and transformed.

The downside of Riemann is that nearly every step of the process requires custom development. Status data and performance metrics must be sent to Riemann, and if software doesn't support this, plugins or scripts need to be written. Ganglia, for instance, supports sending data to Riemann, but this feature is in development and was found to cause instability. The data processing starts as a "blank slate" and must be written before Riemann does anything at all. Which questions need to be answered, and how to answer them must be determined. It also does not analyze previously recorded data. Its configuration is written in Clojure, a dialect of Lisp, which is very different from typical procedural programming and requires time to learn.

3.3 Alerting

The primary goal of a monitoring system is to reduce the occurrence of failures and reduce the time between the failure and its resolution. To do so, administrators must receive timely information that enables them to make decisions about what action must be taken. Administrators should be alerted by the monitoring system in the event that some abnormal condition is detected and may require manual intervention.

Nagios, Zabbix, Sensu, and Riemann all include alerting functionality. This takes the form of email alerting, primarily, but can also include text messaging or alerting via instant messaging and group chat services. The HEPRC group uses email for communication, so all of these systems meet the requirements

in this regard. In addition, **Bosun** [26] and **Cabot** [27], applications designed specifically to generate alerts from monitoring data, were evaluated.

3.3.1 Bosun

Bosun uses "an expressive domain specific language for evaluating alerts and creating detailed notifications" and supports querying Graphite, OpenTSDB, and more recently, InfluxDB [26]. It includes a web-based dashboard which can provide insight into the state of the alerts. The initial setup was complex, and, as with Riemann, configuring alerts required learning a specific language. In addition, using a separate alerting system requires the installation, configuration, and maintenance of another piece of software alongside the metric collection and check scheduling system.

3.3.2 Cabot

Cabot provides the same functionality as Bosun, but does so through a web-based user interface [27]. As such, configuration does not require learning a programming language, but it is more difficult to automate. It only supports Graphite as a data source. As with Bosun, it is an additional piece of software to maintain.

3.4 Visualization

While the goal of this project is to reduce the amount of manual decision-making and intervention, it is desirable to have methods for exploring data visually. In addition, status dashboards allow the discovery of trends and problems that weren't considered when creating alerts. Since real-time charting was a requirement, visualization and dashboarding systems were evaluated. These systems included **Graphite** [28] and **Grafana** [34]. Many such tools exist, but since Graphite is a de-facto standard, it was decided early on to include it regardless of what other choices were made.

3.4.1 Graphite

Graphite is a time-series database and real-time charting application [28]. Any structured numeric data can be sent to Graphite for storage in a database, and arbitrary queries can be run against the stored data. These queries can produce chart or raw data for other purposes. Nearly every monitoring tool discussed has some integration with Graphite, or may rely on Graphite for its data storage. As noted, the decision to include Graphite was made early on.

Graphite consists of a time-series database, Whisper [29], a data caching and aggregating system, Carbon [30], and a web front-end and charting component, Graphite-Web [31]. Data points are stored in Whisper database files in a round-robin fashion, such that new values replace old values after a specific interval [32]. Databases are thus a fixed size that can be determined at the start. Databases can be aggregated for more longer-term storage, as well. It is possible to store values at 1 minute resolution for some duration, and then aggregate these down to 1 hour averages and store them for a much longer duration.

The web front-end allows the data to be explored as a tree, and multiple series can be plotted on a chart. It supports many functions that operate on one or more series, producing derivatives, averages, and sums, for example [33]. The web address of the resulting chart can then be copied and inserted into a web page or email. The user interface is not especially intuitive, and it is primarily used for generating a graph that can then be placed on a dashboard. As such, an additional dashboarding tool is desired.

3.4.2 Grafana

Grafana is a visualization and dashboarding system which supports many data sources, including Graphite, Elasticsearch, and InfluxDB [34]. It provides a much more intuitive user interface than Graphite itself, and allows the easy creation of dashboards that are automatically updated with the latest data. Grafana supports the display of charts, tables, and single numbers in flexible arrangements of panels.

3.5 Automated Remediation

It is common that resolving minor failures or issues in a computing system requires a simple operation, such as restarting a service. While it is desirable that these issues are eliminated in the system itself, in some instances the automated execution of a corrective function is a useful tool. These automated functions may be a simple shell command or a script that makes use of an API.

3.5.1 Sensu

Since Sensu checks and handlers are simply shell commands, a check result can trigger a handler that performs some automated remediation step. Typically checks run on monitored hosts and handlers run on the monitoring server itself, but this can be addressed with a plug-in [35]. This approach is lightweight and integrates well with the rest of the Sensu checks and handlers.

3.5.2 StackStorm

StackStorm provides "event-driven automation" as "it troubleshoots, fixes known problems, and escalates to humans when needed" [36]. It allows complex automation to be triggered by events generated by other systems. With StackStorm, it is possible to connect every piece of software that has an API together. Unlike Sensu, StackStorm is a large and complex system that is typically installed on its own host. Given its power, it has a significant learning curve.

3.6 Log Centralization

Most of the software packages in use on HEPRC servers and VMs generate log files. These log files are crucial for debugging and monitoring, but are scattered across many hosts. Log centralization aims to move this log data to a central server for processing and search. **Elasticsearch** is a database specifically designed for this type of application, and many log centralization systems are built on top of it. These include **Logstash** [38] and **Graylog** [39].

3.6.1 Elasticsearch

Elasticsearch is a data indexing and search engine that allows for real-time analytics and full-text search of schema-less documents. It is not a relational database, but rather a document store with powerful indexing and search capabilities. This makes it ideal for storing and analyzing textual data such as logs.

Logstash and Filebeat are tools that process raw log data into structured documents for storage in Elasticsearch. Filebeat is a lightweight service that runs on hosts generating logs and transmits them to Logstash [40]. Logstash takes the raw log data and processes it using configurable filters and transformations. Figure 2 shows such a processing script and a sample log message. The `grok` filter extracts the specific fields from the log message, while the `date` filter converts the textual date and time into a numerical timestamp with a specific timezone.

Figure 2: A sample log message and Logstash processing script

```
2016-04-20 00:06:37,970 - WARNING - Cleanup - VM Missing a Start attrib on cc-west-↩
    d051c407-f34f-443b-aafa-a89569f618ff.westgrid.

filter {
  if [source] == "/var/log/cloudscheduler.log" {
    grok {
      match => [ "message", "%{TIMESTAMP_ISO8601:timestamp} - %{LOGLEVEL:level} - %{DATA:↩
          task} - %{GREEDYDATA:text}" ]
    }
    date {
      match => [ "timestamp", "yyyy-MM-dd HH:mm:ss,SSS" ]
      timezone => "America/Vancouver"
      remove_field => "timestamp"
    }
  }
}
```

Kibana is a "flexible analytics and visualization platform" that provides "real-time summary and charting of streaming data" [41]. It is a web front-end to Elasticsearch and the data imported by Logstash, Filebeat, Topbeat, and any other source of data. It can be used to explore data through searches, visualize data using complex aggregations and charts, and create dashboards. Kibana was found to be useful for data exploration, but the visualization and dashboarding was much more difficult to configure than Grafana.

The combination of Elasticsearch, Logstash, and Kibana is collectively referred to as the **E.L.K. Stack**. As a whole, it provides a flexible and powerful way to store textual and structured data side-by-side and analyze it. Log centralization only uses a small fraction of the functionality available, but it is an ideal platform for this task.

3.6.2 Graylog

Graylog is built on Elasticsearch, but provides a web-based user interface that is specifically tailored to configuring and maintaining log centralization. Unlike Kibana, Graylog's web interface can be used to fully configure the software. The current release of Graylog is based upon an older release of Elasticsearch, which is not supported by the current release of Kibana. Given the desire to use software that will be continually maintained, this is a downside.

3.7 Tool Evaluation and Selection

The early evaluation involved the use of virtual machines running within Virtualbox (an open source hypervisor) [42] and managed with Vagrant [43] on a single desktop computer. This approach was taken as virtual machines are trivial to create, configure, start, stop, and destroy. Within these virtual machines, the monitoring systems were installed and simple tasks were attempted.

The following packages were selected for this monitoring system:

- **Ganglia** for gathering of performance metrics
- **Sensu** for check execution and monitoring
- **Graphite** for time-series data storage
- **Grafana** for visualization and dashboarding
- **E.L.K. Stack** for log centralization and analysis

These packages provide a nearly complete monitoring system. In addition to these tools, custom software was developed to collect additional data, process it, and display it. This package is called **C'mon Cloud Monitoring** [44] and is described further in the following sections.

4 Cloud Monitoring

The monitoring tools introduced in the previous sections and the C'mon software allow for configurable monitoring of multiple Cloud Scheduler instances. A single monitoring server runs the data processing and storage servers and the visualization tools. Each Cloud Scheduler host runs data collection scripts and checks scheduled by Sensu. This allows the majority of the CPU load caused by processing metrics and other data to be on the monitoring server, and not the hosts performing computationally intensive tasks.

The cloud monitoring system consists of three components: data collection, processing and storage, and visualization. In the sections that follow, the setup and configuration of each component is described.

4.1 Data Collection

The collection of monitoring data is handled by Sensu and Ganglia. Ganglia collects performance data from each VM and each Cloud Scheduler server and sends it to the monitoring server. Sensu executes basic checks, and also schedules the execution of custom data collection scripts every minute. These data provide a nearly complete picture of the status of the clouds.

4.1.1 Sensu

The monitoring server runs the Sensu server, API, and web frontend. Each Cloud Scheduler host runs the Sensu client. Each client must be configured to communicate with the monitoring server. Figure 3 shows the main configuration for the Sensu client running on a Cloud Scheduler host. These clients subscribe to the `cloudscheduler` check group such that they will execute this group of check requests.

Figure 3: Sample Sensu client configuration

```
{
  "rabbitmq": {
    "host": "monitor.heprc.uvic.ca",
    "port": 5672,
    "user": "user_here",
    "password": "password_here",
    "vhost": "/vhost_here"
  },
  "client": {
    "name": "bellecs.heprc.uvic.ca",
    "address": "206.12.154.50",
    "subscriptions": [
      "cloudscheduler"
    ],
    "grid": {
      "name": "Belle-II"
    }
  }
}
```

An sample check configuration is shown in figure 4. This check, named `cloudscheduler-process`, executes the `check-process.rb` script every 60 seconds, which checks for the existence of a process with the name `cloud_scheduler`. It is executed by clients that subscribe to the `cloudscheduler` check group, and if it fails 5 times, the `mailer` handler is executed. While this check does not guarantee that the `cloud_scheduler` process is running properly, it does provide a certain level of assurance.

Figure 4: A sample Sensu check configuration

```
{
  "checks": {
    "cloudscheduler-process": {
      "command": "check-process.rb --pattern cloud_scheduler",
      "subscribers": ["cloudscheduler"],
      "interval": 60,
      "handlers": ["mailer"],
      "occurrences": 5
    }
  }
}
```

The `cloudscheduler-process` check, among others, are executed on each Cloud Scheduler host. When these checks fail a certain number of times (given by `occurrences`), the `mailer` handler is triggered. This handler is shown in figure 5. It operates by executing the `handler-mailer.rb` command with the configuration specified by the `mailer-config` object. This command is provided by the `sensu-plugins-mailer` plugin [45] which can be installed using the command `sensu-install -p mailer`.

Figure 5: Sensu `mailer` handler configuration

```
{
  "handlers": {
    "mailer": {
      "type": "pipe",
      "command": "handler-mailer.rb --content_type plain --json_config mailer_config"
    }
  },
  "mailer_config": {
    "content_type": "plain",
    "admin_gui": "http://monitor.heprc.uvic.ca:3001/",
    "mail_from": "ringda@uvic.ca",
    "mail_to": "ringda@uvic.ca",
    "smtp_address": "smtp.uvic.ca",
    "smtp_port": "25",
    "smtp_domain": "uvic.ca"
  }
}
```

4.1.2 Ganglia

Ganglia data is aggregated and stored by a single `gmetad` running on the monitoring server. For each Cloud Scheduler instance, a `gmond` is also running on the monitoring server on a unique port. The `gmetad` configuration lists each of these `gmond`s in its configuration, shown in figure 6. Since CERN is already making use of Ganglia, this configuration is connecting to their `gmond`s directly for some instances. Data is polled at 15 second intervals and stored at 1 minute resolution for 1 day.

Figure 6: Ganglia `gmetad` configuration

```
data_source "Belle-II" localhost:8660
data_source "IAAS" atlas-ganglia-mon.cern.ch:9004
data_source "Atlas-CERN" atlas-ganglia-mon.cern.ch:8649

RRAs "RRA:AVERAGE:0.5:4:5760"

case_sensitive_hostnames 0
graphite_prefix "ganglia"
carbon_server "localhost"
```

Each Cloud Scheduler host and VM also runs a `gmond` which is configured to send all data to the central `gmond` for that instance. The key parts of this configuration are shown in figure 7.

Figure 7: Ganglia `gmond` configuration

```
cluster {
  name = "Belle-II"
  owner = "unspecified"
  latlong = "unspecified"
  url = "unspecified"
}

udp_send_channel {
  host = monitor.heprc.uvic.ca
  port = 8660
  ttl = 1
}
```

4.1.3 C'mon Collectors

The C'mon data collection scripts extract raw data from Cloud Scheduler using the `cloud_status` command, and HTCondor using the `htcondor` Python package. They do not perform any processing of the data, but instead send the raw data to the monitoring server over Sensu's message queue, RabbitMQ. This requires the Pika Python package [46], but otherwise there are no dependencies.

There are two data collection scripts: `cmon-collect-status.py`, and `cmon-collect-resources.py`. The `cmon-collect-status.py` script queries HTCondor and Cloud Scheduler for general status using shell commands and APIs. The `cmon-collect-resources.py` script makes use of Cloud Scheduler internals to connect to each cloud directly and retrieve a full list of running VMs. This allows any differences between Cloud Scheduler's list of VMs and the actual list from each cloud to be discovered.

A portion of the `cmon-collect-status.py` script is shown in figure 8. This code is responsible for connecting to the RabbitMQ server and sending a message payload containing the raw status information.

Figure 8: `cmon-collect-status.py` message send code

```
payload = {
    'grid': grid_name,
    'clouds': clouds,
    'jobs': jobs,
    'slots': slots,
}
payload = json.dumps(payload)

creds = pika.PlainCredentials(RMQ_USER, RMQ_SECRET)
params = pika.ConnectionParameters(RMQ_SERVER, RMQ_PORT, RMQ_VHOST, creds)
rmq = pika.BlockingConnection(params)

props = pika.BasicProperties(
    delivery_mode=2,
    timestamp=int(time.time())
)

channel = rmq.channel()
channel.exchange_declare(exchange='cmon', exchange_type='fanout')
channel.basic_publish(exchange='cmon', routing_key='', body=payload, properties=props)

rmq.close()

print "OK: %d bytes of JSON sent for %s" % (len(payload), grid_name)
```

4.2 Processing and Storage

The raw data from the Cloud Scheduler servers is processed by a separate script called `cmon` which runs as a daemon. It connects to RabbitMQ, and stores data in Graphite and MongoDB [47]. As new data becomes available, the data stored in MongoDB is updated, and a history of changes is maintained.

MongoDB was selected as the data storage back-end because it allows for structured but schema-less data. Any arbitrary structured data can be stored and queried by MongoDB, much like Elasticsearch. This is in contrast to a relational database where each field to be stored must be predefined. It supports powerful and expressive aggregations and querying using JavaScript, and has a client library for Python.

In the `cmon` database, there are five collections: `clouds`, `grids`, `jobs`, `vms`, and `status`. The `clouds` collection lists each cloud that is configured in each Cloud Scheduler host and some basic parameters. The `grids` collection stores a numerical snapshot of each Cloud Scheduler hosts's status, and is used to generate the status summary shown in figure 12. The `jobs` and `vms` collections list every job and VM in every Cloud Scheduler and HTCondor host and maintains a history of each. Finally, the `status` collection stores the most recent raw data.

Each message sent by the collection scripts is timestamped and sent over a persistent message queue. This means that if the `cmon` daemon is not running for some reason, it can process the data it missed when it is next online. This ensures that there are no gaps in the historical data.

4.2.1 Graphite

Graphite stores time-series data generated by Ganglia and C'mon. The `gmetad` configuration in figure 6 sends all metric data to Graphite, and C'mon generates summary metrics such that, for example, the number of running jobs can be plotted over the past week.

Figure 9: Graphite storage schemas for Ganglia and C'mon

```
[grids]
pattern = ^grids\..*
retentions = 1m:30d, 10m:1y, 1d:5y

[ganglia]
pattern = ^ganglia\.
retentions = 1m:7d, 15m:30d
```

Graphite is configured as shown in figure 9 to store C'mon data at 1 minute intervals for 30 days, 10 minute averages for 1 year, and 1 day averages for 5 years, and Ganglia data at minute intervals for 7 days and 15 minute averages for 30 days. This allows data to be viewed with high resolution when it is valuable, and maintains historical information for comparison and analysis. Keeping high resolution data for long periods of time is wasteful of disk space and generally not helpful. More important is allowing immediate problems to be identified and addressed.

4.3 Visualization

As noted, Grafana was selected as the visualization component for this project. It was found that Grafana was not ideal for generating dense tabular displays, which are useful for providing a status overview. Figure 10 shows a portion of the existing cloud monitoring page in use by the HEPRC group, which is text-based and generated by each server.

```
ATLAS-CS              Sta  Run  Ret  Err      Condor-slots                          Idle-VMs
cern-atlas-1c           0   93    0    0      93  93  93   4   0   0   0   0         0
cern-atlas-8c           5    2    0    0       1                                    1
Jobs - All     (Tot, Que, Run, Hel) =   312    29   283     0
Jobs - MCore   (Tot, Que, Run, Hel) =     1     0     1     0
Jobs - Himem   (Tot, Que, Run, Hel) =   308    29   279     0
Jobs - Analy   (Tot, Que, Run, Hel) =     3     0     3     0
Jobs - DPHEP   (Tot, Que, Run, Hel) =     0     0     0     0
Tue Apr 26 21:46:19 CEST 2016
```

Figure 10: Text-based cloud monitoring report

While this monitoring page provides a summary of the status, it does not provide any historical information or charting, nor access to more detailed information. At first a similar approach was attempted using Grafana, however it proved difficult to achieve the same level of data density. Grafana is better suited to large charts and small tables. Instead, a web frontend to C'mon was developed.

4.3.1 C'mon Web

The C'mon web frontend is built upon the Flask microframework for Python [48]. It simplifies the development of small web applications while still allowing the use of the full power of Python. Using the PyMongo package [49], this application communicates with the MongoDB database and retreieves summary and detailed information from the collections described in section 4.2 on request.

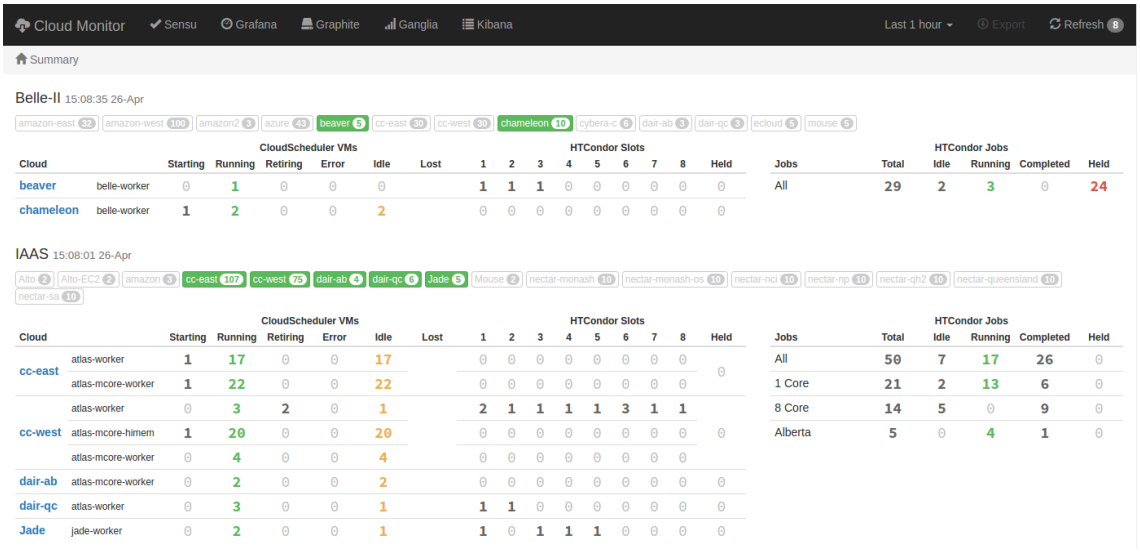


Figure 11: C'mon web frontend summary page

Figure 11 shows the C'mon web frontend summary page. This page lists each Cloud Scheduler host and all of its running clouds, VMs, and jobs. Figure 12 shows the cloud list more closely. Beside each cloud is a count of VMs in the starting, running, retiring, error, or idle states. Figure 13 provides a closer view of the jobs list with each job type and a count of jobs in the idle, running, completed, and held states, along with a total.

		CloudScheduler VMs						HTCondor Slots								
Cloud		Starting	Running	Retiring	Error	Idle	Lost	1	2	3	4	5	6	7	8	Held
cc-east	atlas-worker	1	27	0	0	27		0	0	0	0	0	0	0	0	0
	atlas-mcore-worker	0	19	0	0	19		0	0	0	0	0	0	0	0	
cc-west	atlas-worker	0	18	1	0	0		19	19	19	19	19	18	18	15	0
	atlas-mcore-worker	0	48	0	0	0		48	0	0	0	0	0	0	0	
dair-ab	atlas-worker	0	2	0	0	0		2	2	2	2	2	1	1	0	0
	atlas-mcore-worker	0	2	0	0	0		2	0	0	0	0	0	0	0	
dair-qc	atlas-worker	0	3	0	0	0		3	3	3	3	3	3	0	0	0
	atlas-mcore-worker	0	3	0	0	0		3	0	0	0	0	0	0	0	
Jade	jade-worker	0	2	0	0	1		1	1	1	0	0	0	0	0	0

Figure 12: Closer view of a C'mon clouds list

HTCondor Jobs

Jobs	Total	Idle	Running	Completed	Held
All	56	0	17	39	0
1 Core	27	0	12	15	0
8 Core	15	0	2	13	0
Alberta	4	0	3	1	0

Figure 13: Closer view of a C'mon jobs list

Clicking on a numerical value in either list will cause it to be plotted, as shown in figure 14. Multiple values can be plotted simultaneously, and a legend automatically appears when more than one value is plotted. These plots are generated using the Plotly.js library [50], which provides powerful, high performance, interactive charts in a web browser.

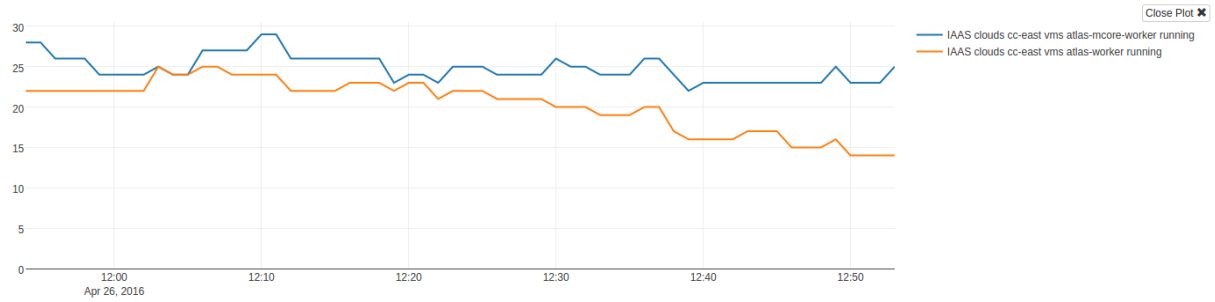
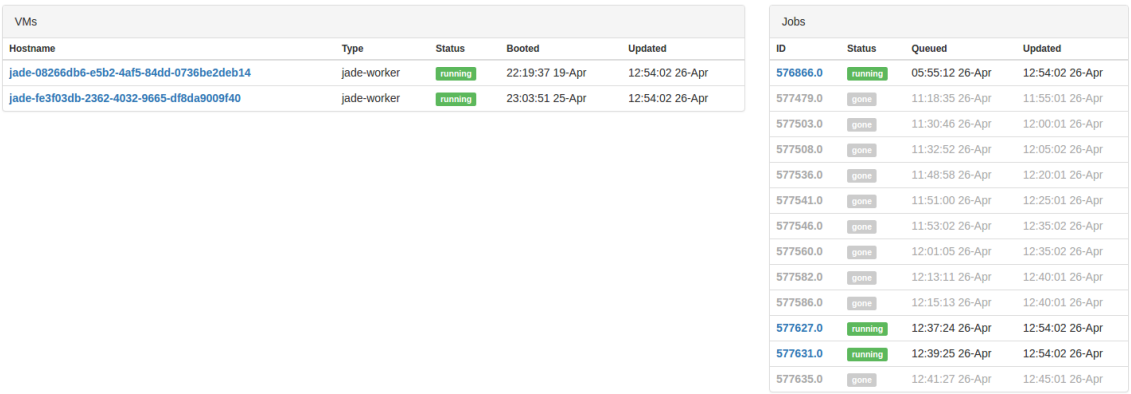


Figure 14: C'mon plotting multiple values over time

In addition, clicking on the name of any cloud will lead to a cloud details page as shown in figure 15. This page lists each VM and job that Cloud Scheduler and HTCondor have had information about in the past hour. The hostname, type, status, and time last updated is shown for each VM, and the status and time queued and updated is shown for each job.



VMs

Hostname	Type	Status	Booted	Updated
jade-08266db6-e5b2-4af5-84dd-0736be2deb14	jade-worker	running	22:19:37 19-Apr	12:54:02 26-Apr
jade-fe3f03db-2362-4032-9665-df8da9009f40	jade-worker	running	23:03:51 25-Apr	12:54:02 26-Apr

Jobs

ID	Status	Queued	Updated
576866.0	running	05:55:12 26-Apr	12:54:02 26-Apr
577479.0	gone	11:18:35 26-Apr	11:55:01 26-Apr
577503.0	gone	11:30:46 26-Apr	12:00:01 26-Apr
577508.0	gone	11:32:52 26-Apr	12:05:02 26-Apr
577536.0	gone	11:48:58 26-Apr	12:20:01 26-Apr
577541.0	gone	11:51:00 26-Apr	12:25:01 26-Apr
577546.0	gone	11:53:02 26-Apr	12:35:02 26-Apr
577560.0	gone	12:01:05 26-Apr	12:35:02 26-Apr
577582.0	gone	12:13:11 26-Apr	12:40:01 26-Apr
577586.0	gone	12:15:13 26-Apr	12:40:01 26-Apr
577627.0	running	12:37:24 26-Apr	12:54:02 26-Apr
577631.0	running	12:39:25 26-Apr	12:54:02 26-Apr
577635.0	gone	12:41:27 26-Apr	12:45:01 26-Apr

Figure 15: C'mon cloud details page showing a list of VMs and jobs

From the cloud details page, clicking on any VM hostname or job ID will lead to a further page with additional details and historical information about that VM or job. An example VM details page is shown in figure 16. It includes the time of each status change, and a list of all jobs that have been run on the VM in

the past hour along with their current state. In addition, Elasticsearch is being queried for log information regarding the VM or job.

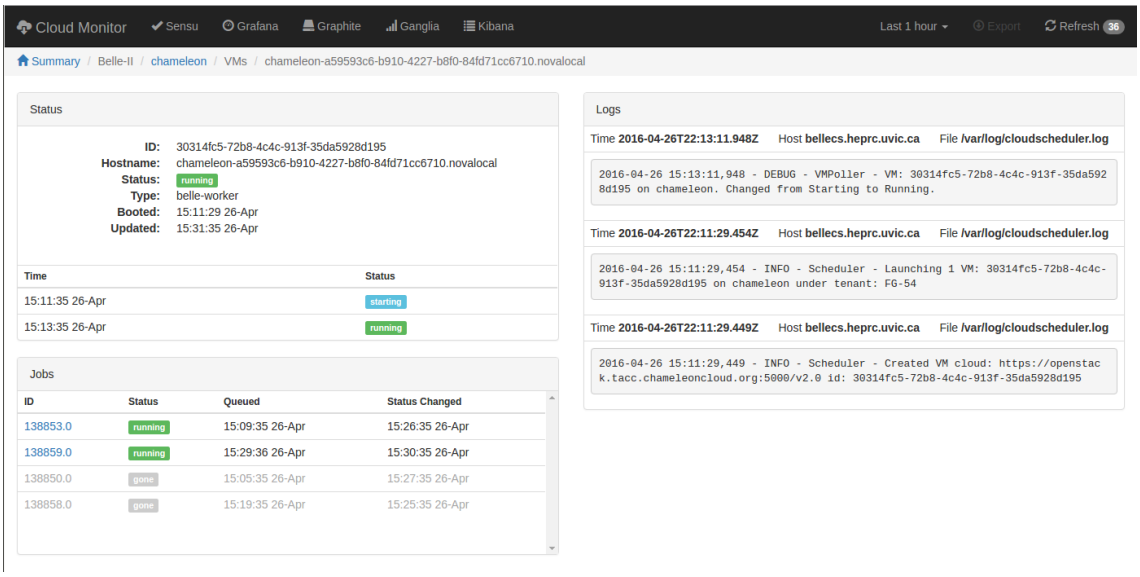


Figure 16: C'mon VM details showing status, history, and logs

4.4 Deployment

During the evaluation, development, and testing of the monitoring system, packages were installed manually, and configuration files were created, modified, and removed manually. While is this is a normal development practice, it is fragile in a production environment. An administrator needs to know about each package that is being used, where its configuration files are located, and how the various components interact. By automating the deployment and configuration using Ansible [51], all of this information is collected in one place and can be easily understood.

Ansible is described as a "simple IT automation engine that automates cloud provisioning, configuration management, application deployment, intra-service orchestration, and many other IT needs" [51]. It is capable of running a series of *tasks* grouped into *roles*, known as a *playbook*, across multiple hosts. These tasks can be used to install software packages, generate configuration files, manage services, and execute any arbitrary script, among many other things. This allows the configuration and deployment of servers to be centralized and automated.

Ansible executes tasks which are grouped into roles. The monitoring configuration consists of two roles: `monitor` and `cloudscheduler`. The `monitor` role is applied to a single centralized monitoring server, while the `cloudscheduler` role is applied to the servers to be monitored. Both roles install and configure the various components of the monitoring system previously discussed.

A short excerpt from the monitoring system Ansible playbook is shown in figure 17. The task, written in YAML [52], copies multiple configuration templates to the server and processes them. Figure 18 is a sample template which makes use of Ansible *facts* about a host (in this case its domain name, `ansible_fqdn` and IP address, `ansible_default_ipv4.address`). Wrapping these identifiers in {{ }} causes them to be replaced with the values when the template is processed.

Figure 17: Ansible playbook excerpt from the `monitor` role

```
- name: configure sensu checks and handlers
  template: src=sensu/conf.d/{{ item }}.j2 dest=/etc/sensu/conf.d/{{ item }} owner=root ↩
      group=root mode=0644
  with_items:
    - checks-cloudmonitor.json
    - checks-cloudscheduler.json
    - checks-condor.json
    - checks-ganglia.json
    - checks-graphite.json
    - handler-mailer.json
  tags: [config,sensu]
```

Figure 18: Sample configuration template, `client.json.j2`, from an Ansible playbook

```
{
  "client": {
    "name": "{{ ansible_fqdn }}",
    "address": "{{ ansible_default_ipv4.address }}",
    "subscriptions": [
      "monitor"
    ]
  }
}
```

In all, the Ansible playbook for the `monitor` role executes 62 tasks, installing 28 packages and their dependencies and generating 29 configuration files from templates. The `cloudscheduler` role executes 17 tasks, installing 4 packages and their dependencies and generating 9 configuration files from templates.

5 Conclusion

Given the dynamic nature of cloud computing, a monitoring system for these clouds must be equally dynamic. Many monitoring tools have been evaluated, and many were found to meet the needs of traditional computing but lacked key features required for this application. Instead, newer monitoring platforms were found to be better able to adapt to this environment. In addition, some custom development work was required to integrate these components and create a system that will allow the HEPRC group to build from. Finally, a method for automated deployment and configuration was implemented.

6 Future Work

Though InfluxDB and the other components of that stack are still very new, it is a platform with considerable promise. It is recommended that these tools be reviewed and re-evaluated in 1-2 years. In addition, the custom monitoring scripts could be refactored into a library of monitoring functions specifically for HEP cloud computing applications.

Integrating more monitoring functions directly into Cloud Scheduler and HTCondor would also serve to simplify the monitoring system itself. These services could periodically report data to a centralized monitoring server in some configurable way. Since these systems already have internal representations of their current state, this would eliminate the overhead of extracting that state through commands and APIs, parsing and processing it, and then storing it.

Finally, more focus should be put on automatic remediation now that this monitoring system is in place. Email alerting is required so that administrators are aware of issues as they arise, but corrective measures

should be automated where possible. This automation can be in the form of corrective handlers in Sensu, or through modifications to the monitored services themselves.

7 Glossary

API Application Programming Interface

Cloud Scheduler - manages cloud VMs for HEP jobs.

daemon - a background process, typically a server.

HTCondor - batch job scheduler.

JSON JavaScript Object Notation - a textual format for data serialization based on JavaScript's notation for object literals.

VM Virtual Machine - emulation or virtualization of a computer system.

YAML YAML Ain't Markup language - a textual format for data serialization.

References

[1] High Energy Physics Research Computing. (2016) Home — High Energy Physics Research Computing [Online]. Available: http://heprc.phys.uvic.ca/home [Apr. 25, 2016].

[2] High Energy Physics Research Computing. (2016, Feb.) Cloud Scheduler GitHub repository [Online]. Available: https://github.com/hep-gc/cloud-scheduler [Apr. 26, 2016].

[3] Computer Sciences Department, University of Wisconsin. (2016) HTCondor - Home [Online]. Available: https://research.cs.wisc.edu/htcondor/ [Apr. 26, 2016].

[4] J. Olson *et al.* (2015, Sep.) Munin [Online]. Available: http://munin-monitoring.org/ [Apr. 26, 2016].

[5] High Energy Physics Research Computing. (2016, Feb.) Munin Collector GitHub repository [Online]. Available: https://github.com/hep-gc/munin-collector [Apr. 26, 2016].

[6] High Energy Physics Research Computing. (2016, Jan.) Monitoring Requirements & Design [Online]. Available: https://wiki.heprc.uvic.ca/twiki/bin/view/HEPrcPrivate/MonitorReqs [Apr. 25, 2016].

[7] The Ganglia Project. (2015, Apr.) Ganglia Monitoring System [Online]. Available: http://ganglia.info/ [Apr. 19, 2016].

[8] C. Cordeiro. (2014, Mar.) Ganglia Hierarchical Deployment [Online]. Available: https://twiki.cern.ch/twiki/bin/view/ArdaGrid/GangliaHierarchicalDeployment [Apr. 19, 2016].

[9] CERN (2016, Apr.) CernVM — cernvm.web.cern.ch [Online]. Available: https://cernvm.cern.ch/ [Apr. 26, 2016].

[10] Influx Data. (2015) The Platform for Time-Series Data — InfluxData. Available: https://influxdata.com/ [Apr. 25, 2016].

[11] Influx Data. (2016, Apr.) Telegraf Changelog [Online]. Available: https://github.com/influxdata/telegraf/blob/master/CHANGELOG.md [Apr. 25, 2016].

[12] Influx Data. (2015) Chronograf – Time-Series Data Visualization — InfluxData. Available: https://influxdata.com/time-series-platform/chronograf/ [Apr. 25, 2016].

[13] Influx Data. (2015) Kapacitor – Time Series Data Processing, Monitoring, & Alerting — InfluxData. Available: https://influxdata.com/time-series-platform/kapacitor/ [Apr. 25, 2016].

[14] Elasticsearch BV. (2016) Topbeat — Elastic [Online]. Available: https://www.elastic.co/products/beats/topbeat [Apr. 25, 2016].

[15] Nagios Enterprises. (2016) Nagios - The Industry Standard In IT Infrastructure Monitoring [Online]. Available: https://www.nagios.org/ [Apr. 20, 2016].

[16] Stack Exchange Inc. (2009) monitoring - What tool do you use to monitor your servers? [Online]. Available: http://serverfault.com/questions/44/what-tool-do-you-use-to-monitor-your-servers [Apr. 20, 2016].

[17] Zabbix LLC. (2016) Zabbix :: The Enterprise-Class Open Source Network Monitoring Solution [Online]. Available: http://www.zabbix.com/ [Apr. 22, 2016].

[18] Zabbix LLC. (2016) Active agent auto-registration [Zabbix Documentation 3.2] [Online]. Available: https://www.zabbix.com/documentation/3.2/manual/discovery/auto_registration

[19] Heavy Water Operations, LLC. (2016) Sensu — Monitoring for today's infrastructure [Online]. Available: https://sensuapp.org/ [Apr. 26, 2016].

[20] S. Porter. (2011, Jul.) Sensu GitHub commit [Online]. Available: https://github.com/sensu/sensu/commit/c5c8071bb37fcf359606bced92035e3f4ac1e0f3 [Apr. 25, 2016].

[21] S. Sanfilippo. (2016) Redis [Online]. Available: http://redis.io/ [Apr. 26, 2016].

[22] Pivotal Software, Inc. (2016, Mar.) RabbitMQ - Messaging that just works [Online]. Available: [Apr. 26, 2016].

[23] Y. Matsumoto. (2016, Apr.) Ruby Programming Language [Online]. Available: https://www.ruby-lang.org/en/about/ [Apr. 26, 2016].

[24] Ecma International, The JSON Data Interchange Format, *ECMA-404*, s.l.: Ecma International Oct. 2013. [Online]. Available: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf [Apr. 26, 2016].

[25] K. Kingsbury. (2016) Riemann - A network monitoring system [Online]. Available: http://riemann.io/ [Apr. 22, 2016].

[26] Stack Exchange, Inc. (2016) Bosun [Online]. Available: http://bosun.org/ [Apr. 26, 2016].

[27] Arachnys Informaton Services Ltd (2016) Cabot - monitor and alert [Online]. Available: http://cabotapp.com/ [Apr. 26, 2016].

[28] Graphite Project. (2012, May) Graphite - Scalable Realtime Graphing [Online]. Available: http://graphite.wikidot.com/ [Apr. 26, 2016].

[29] Graphite Project. (2016, Apr.) Whisper GitHub repository [Online]. Available: https://github.com/graphite-project/whisper [Apr. 26, 2016].

[30] Graphite Project. (2016, Apr.) Carbon GitHub repository [Online]. Available: https://github.com/graphite-project/carbon [Apr. 26, 2016].

[31] Graphite Project. (2016, Apr.) Graphite-Web GitHub repository [Online]. Available: https://github.com/graphite-project/graphite-web [Apr. 26, 2016].

[32] Graphite Project. (2015, June) Configuring Carbon - Graphite 0.10.0 documentation [Online]. Available: http://graphite.readthedocs.org/en/latest/config-carbon.html#storage-schemas-conf [Apr. 26, 2016].

[33] Graphite Project. (2013, Mar.) Functions - Graphite 0.10.0 documentation [Online]. Available: http://graphite.readthedocs.org/en/latest/functions.html [Apr. 26, 2016].

[34] T. degaard and Coding Instinct AB. (2016) Grafana - Graphite and InfluxDB Dashboard and graph composer [Online]. Available: http://grafana.org [Apr. 26, 2016].

[35] Sensu-Plugins and contributors. (2016, Apr.) Sensu-Plugins-Sensu GitHub repository [Online]. Available: https://github.com/sensu-plugins/sensu-plugins-sensu [Apr. 26, 2016].

[36] StackStorm, Inc. (2016) StackStorm — Event-driven automation [Online]. Available: https://stackstorm.com/ [Apr. 26, 2016].

[37] Elasticsearch BV. (2016) Elasticsearch — Elastic [Online]. Available: https://www.elastic.co/products/elasticsearch [Apr. 26, 2016].

[38] Elasticsearch BV. (2016) Logstash — Elastic [Online]. Available: https://www.graylog.org/ [Apr. 26, 2016].

[39] Graylog, Inc. (2016) Graylog — Open Source Log Management [Online]. Available: https://www.elastic.co/products/logstash [Apr. 26, 2016].

[40] Elasticsearch BV. (2016) Filebeat — Elastic [Online]. Available: https://www.elastic.co/products/beats/filebeat [Apr. 27, 2016].

[41] Elasticsearch BV. (2016) Kibana: Explore, Visualize, Discover Data — Elastic [Online]. Available: https://www.elastic.co/products/kibana [Apr. 22, 2016].

[42] Oracle Corporation (2016, Apr.) Oracle VM VirtualBox [Online]. Available: https://www.virtualbox.org/ [Apr. 26, 2016].

[43] HasiCorp. (2016) Vagrant by HashiCorp [Online]. Available: https://www.vagrantup.com/ [Apr. 26, 2016].

[44] D. Ring. (2016, Apr.) C'mon Cloud Monitoring GitHub repository [Online]. Available: https://github.com/hep-gc/cloud-monitoring [Apr. 26, 2016].

[45] Sensu-Plugins and contributors. (2016, Apr.) Sensu-Plugins-Mailer GitHub repository [Online]. Available: https://github.com/sensu-plugins/sensu-plugins-mailer [Apr. 26, 2016].

[46] T Garnock-Jones, *et al.* (2015, Apr.) Pika GitHub repository [Online]. Available: https://github.com/pika/pika/tree/master [Apr. 26, 2016].

[47] MongoDB, Inc. (2016) MongoDB for GIANT Ideas — MongoDB [Online]. Available: https://www.mongodb.org/ [Apr. 26, 2016].

[48] A. Ronacher. (2013, June) Welcome — Flask (A Python Microframework) [Online]. Available: http://flask.pocoo.org/ [Apr. 26, 2016].

[49] MongoDB, Inc. (2016) PyMongo 3.2.2 Documentation [Online]. Available: https://api.mongodb.org/python/current/ [Apr. 26, 2016].

[50] Plotly, Inc. (2016, Apr.) plotly.js — JavaScript Graphing Library [Online]. Available: https://plot.ly/javascript/ [Apr. 26, 2016].

[51] Red Hat, Inc. (2016) How Ansible Works — Ansible.com [Online]. Available: https://www.ansible.com/how-ansible-works [Apr. 19, 2016].

[52] O. Ben-Kiki, C. Evans, I. dt Net. (2009, Oct.) YAML Ain't Markup Language (YAML^{TM}) Version 1.2 [Online]. Available: http://www.yaml.org/spec/1.2/spec.html [Apr. 26, 2016].