EPJ Web of Conferences **251**, 02047 (2021) https://doi.org/10.1051/epjconf/202125102047
CHEP 2021

A hybrid system for monitoring and automated recovery at the Glasgow Tier-2 cluster

Emanuele Simili[1,*], *Gordon* Stewart[1], *Gareth* Roy[1], *Samuel* Skipsey, and *David* Britton[1]

[1]School of Physics and Astronomy, University of Glasgow
Kelvin Building, University Avenue, G12 8QQ, United Kingdom

Abstract. We have deployed a central monitoring and logging system based on Prometheus, Loki and Grafana that collects, aggregates and displays metrics and logs from the Tier-2 ScotGrid cluster at Glasgow. Bespoke dashboards built on Prometheus metrics give a quick overview of cluster performance and make it easy to identify issues. Logs from all nodes and services are collected to a central Loki server and retained over time. This integrated system provides a full overview of the cluster's health and has become an essential tool for daily maintenance and in the investigation of any issue.
The system includes an automated alerting application that parses metrics and logs and can send notifications when specified conditions are met, and as a further step toward automation, can also perform simple recovery actions based on well known issues and their encoded solutions. The general purpose is to create a more resilient Tier-2 cluster where human intervention is kept to a minimum. Given the funding constraints experienced by many academic research institutions, this promises to free staff from routine tasks, allowing them to address their expertise to more interesting problems.
In this paper, we describe the tools and set-up of the existing monitoring system, the automated recovery methods implemented so far, and the plan for further automation.

1 Introduction

Along with the monitoring from experiments and the accounting tools in use on the WLCG, Grid sites implement their own solutions to monitor local resources, such as computing, storage and networking.

The general purpose of monitoring is to concisely pinpoint issues by reporting key hardware and service metrics, as well as detailed outputs and logs. Automated alerting tools based on system metrics already exist, but they generally do little more than push out messages notifying a sys-admin of the need for manual intervention. The normal work-flow for a sys-admin is, therefore, to watch dashboards and logs, to identify and diagnose problems, and to take appropriate remedial action. Some of these actions may be fairly simple (such as rebooting a problematic server or cleaning up a full disk partition) and can be easily done remotely by taking advantage of the configuration management tools widely used by any Grid site.

Following some general ideas that were outlined in our previous work [1], we have developed a hybrid control system that can automate the work-flows for a number of simple, well

*e-mail: emanuele.simili@glasgow.ac.uk

known use-cases; the solutions employed in these cases are encoded in a set of commands or a script.

2 Site Description

The Glasgow Tier-2 cluster comprises a standard Grid submission system with over 6000 job slots, another 800 job slots in VAC [2] machines, and about 4 PB of Ceph [3] based storage. Figure-1 provides a simplified view of the Glasgow Tier-2 cluster. The job submission path is outlined, as well as the authentication mechanism and data channels.

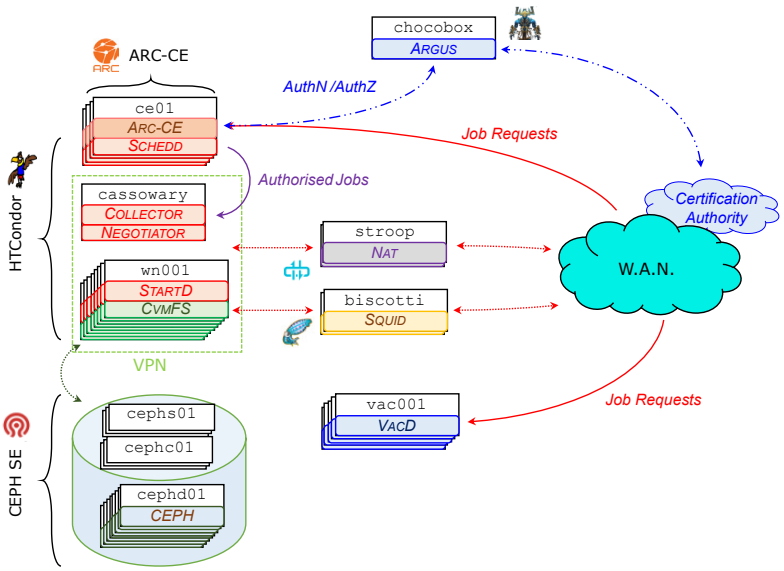

Figure 1. Schematic view of the Glasgow Tier-2 cluster. Solid lines represent the path of submitted jobs, dash-dotted lines represent the authentication chain, and dotted lines represent data communication channels.

The standard submission system consists of a few hundred[1] worker-nodes managed by NorduGrid ARC [4], ARGUS [5] and HTCondor [6]. It provides computing power to LHC experiments including ATLAS and CMS, as well as to other non-LHC experiments such as LIGO and CLAS12. Jobs are submitted by WLCG users and experiments to four ARC Compute Elements (ARC-CEs), which are either physical or virtual machines with dual network interfaces (`ce01`, `ce02`, ...); in turn, these CEs submit jobs to the underlying HTCondor batch system for distribution among the physical worker-nodes (`wn001`, `wn002`, ...).

Authentication and authorization of incoming jobs is performed by an ARGUS server (`chocobox`), running on a dedicated virtual machine. Once approved, jobs are passed by the ARC-CE to the workload management system (HTCondor) scheduler where they are queued for execution by the batch system. The batch system is overseen by another virtual machine (`cassowary`) which runs the HTCondor manager; this collects scheduled jobs and, when the necessary resources become available, assigns them to the actual batch system that executes them on one of the worker-nodes. Jobs can read and write data to and from the local Storage Element (SE), or to and from the internet via two NAT servers (`stroop` and `wafel`) and two

[1]The site is currently at reduced capacity due to an on-going relocation to a new data centre.

redundant Squid systems (`biscotti` and `cantucci`), which act as a cache for the CernVM file-system (CVMFS [7]).

Beside the conventional Grid submission system, the Glasgow site also runs a smaller number of VAC machines [2] (`vac00`1, `vac00`2, ...) to provide an Infrastructure-as-a-Service (IaaS) cloud for various other experiments, including LHCb.

The Storage Element is composed of three sets of physical machines running Ceph [3], including about 20 dedicated storage machines or disks (`cephd0x`), six physical caches (`cephc0x`) and three entry points or servers (`cephs0x`).

For simplicity, other essential components of the site – including networking, provisioning and configuration management, and the Grid information system (BDII) – are not depicted in Figure-1. One server particularly worthy of note is our central management server `foithong`, on which Ansible [8] is installed together with the complete set of roles and playbooks necessary to install and configure the various components of the cluster, as discussed in the following section.

2.1 Remote Management Tools

Due to the size and complexity of the cluster, it is crucial to rely on a central management tool to automate most of the steps needed to turn a bare metal (or bare virtual) machine into a fully-operational worker-node or service provider.

Once the minimal operating system has been established, all additional software is installed and configured with Ansible [8]. We have invested much time developing a complete set of roles and playbooks that covers every component of the cluster. For safety, our configuration management system is version-controlled and regularly backed-up to our internal Git [9] repository.

Ansible is not only a configuration management tool, but it can also be used to remotely execute recovery actions, as long as such actions can be encoded in a series of remote commands within an Ansible playbook. With this in mind, we have developed a library of Ansible playbooks for regular maintenance tasks, such as cleaning up CVMFS caches, enabling or disabling HTCondor on individual worker-nodes, and adding the pool accounts and other configuration necessary to support a new Virtual Organization (VO). The simplicity of executing recovery actions from a single access point fits very well with the superior level of automation we wish to achieve.

3 Monitoring and Logging

Metrics offer an instantaneous measurement of the state, or the change in state, of the hardware and software environment on a machine. Metrics are numerical values, either integer or floating point numbers, associated with a specific measurement. Logs are textual and contain the messages generated by running services.

For the development of our monitoring solution, we looked for a platform that would allow the aggregation of metrics and logs under a common framework, to provide a better overview of the cluster's state and to help us take educated decisions based on all available intelligence. After examining various commercial and open-source platforms, we opted to employ the Prometheus-Loki-Grafana (PLG) software stack [10] [11] [12].

The monitoring system involves a client service (*node_exporter* [13]) that runs in the background and periodically collects and exports metrics to a dedicated server (`gingersnap`) running Prometheus [10]; Prometheus aggregates the metrics and makes them available to be queried. The Prometheus instance can run on a virtual machine with only a few processor

cores, but it requires a significant amount of RAM as the task of aggregating metrics from hundreds of nodes is quite demanding.

The centralized logging system involves a client service (*PromTail* [14]) that exports system logs to a dedicated server (`victoria`) running Loki [11]. Loki labels and aggregates logs within a database, which is retained for a configurable period of time. The memory requirements of Loki are slightly lower than those of Prometheus, but a large amount of storage is needed as the log output from hundreds of nodes can exceed several GB per day. In our case, storage needs have been met by the availability of a disk server that had been retired from our storage cluster.

We use a separate server (`speculaas`) running Grafana [12], to query and visualize the data. Grafana is a light-weight service that can be installed on a small virtual machine. It provides a common web interface for data display, dashboard management and log querying. Another essential component of the set-up is the Prometheus tool AlertManager [15], which sends alerts when a given set of conditions is met, and which can trigger external actions via the AM-Executor [16] component (this part will be further discussed in Section-4). Finally, long-term storage of metrics is catered for by the VictoriaMetrics [17] tool, which we have installed on the same server as Loki (`victoria`) to take full advantage of the large amount of storage available.

Figure-2 illustrates the flow of metrics and logs between the interconnected services in our monitoring system.

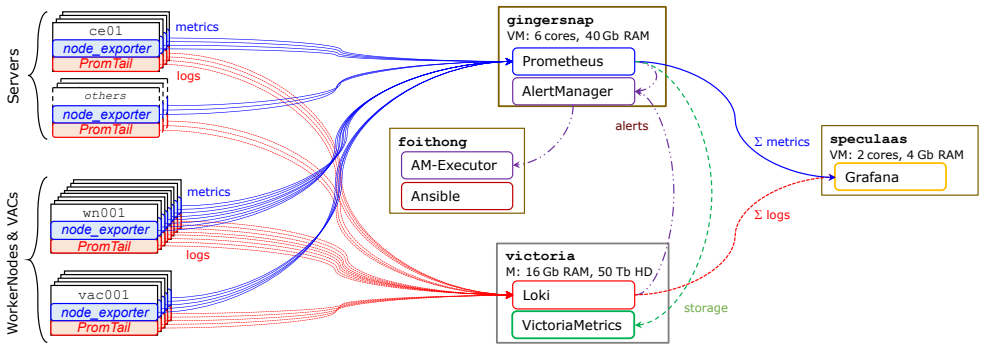

Figure 2. Schematic view of the interconnected systems responsible for monitoring and logging, and the data path between them (see text for further description). Each rectangular block represents a server, and inner rounded blocks represent running services.

3.1 Metric Exporter

To export metrics from each machine we use node_exporter [13], the standard client tool for Prometheus. Ideally, node_exporter would be installed on every machine in the cluster, and must publish metrics to a network port reachable by the Prometheus server. The service is light enough to show no visible impact on CPU and network performance; it produces numerous metrics by default (such as CPU load, network activity, and disk I/O rates), and it can be further customised with ad-hoc metrics.

With a little extra work, we have implemented a number of scripts that extend the output of node_exporter to include batch system statistics and hardware-specific metrics such as temperature readings. In particular, we have added the following capabilities:

- **Temperatures:** Hard-disk temperature is collected by the S.M.A.R.T. [18] tool *hddtemp*, while the ambient temperature is retrieved from motherboard sensors by *ipmitools* [19].

- **HTCondor statistics:** Job statistics are collected on the ARC-CEs and HTCondor manager by a custom call to the *condor_status* command. Job slot usage is collected at node-level via a similar mechanism.

- **VAC information:** Statistical information about the number and status of virtual machines (each containing a job) running across our VAC estate.

- **Process statistics:** Information about running processes is collected using standard Linux commands (such as *ps*) and properly parsed to extract values such as run-time and resource usage.

- **Number of reboots:** A static counter is incremented at every boot to keep track of occasions when a node restarts unexpectedly, as such occurrences can be indicative of hardware failure. Up-time from individual machines is also exported by default.

- **Static information:** Information about the system's hardware (such as number of cores, amount of RAM, and arrangement of physical disks) is collected by a script which runs at boot and uses standard Linux commands (such as *lscpu*, *vmstat* and *lsblk*). The purpose of these metrics, which overlap to a degree with the standard output of node_exporter, is to detect changes in hardware configuration caused by hardware failure, such as the loss of a bank of memory or a processor. It also provides the base-line from which we normalize CPU load and RAM usage.

These scripts run periodically as cron jobs and write metrics to a specific directory. All metrics are then exported to a single endpoint by node_exporter (to TCP port 9100). Prometheus scrapes these metrics and aggregates them into a time-series database, which is made available to Grafana for visualization and querying (see Section-5 for further details). The retention time for metrics is set to one month. After this period has elapsed, metrics are shipped to the VictoriaMetrics server where they are pruned and kept for a further six months.

3.2 Central Logging

To export logs we use PromTail [14], an agent which tails (i.e. extracts recent entries from) the content of local log files and sends this content to a central server running Loki. We have chosen this approach because a single agent is sufficient to push out all the logs to a single end-point (TCP port 9080), and as a Grafana utility, Loki integrates well with the rest of our system. Ideally, PromTail is installed on every machine in the cluster and must be reachable by the Loki server.

Unlike node_exporter, which generates a default set of metrics, PromTail only scrapes the logs that are specified in its configuration, which must be written to take into account both default logs (such as journal and system messages) and custom logs specific to each service.

We have tuned the configuration of PromTail to the various machine types present in our cluster, as shown in table 1. Ceph's logs are not mentioned in this table, as our storage is monitored separately.

Table 1. Custom PromTail configurations for the main services except storage.

Service	Exported Logs
all	/var/log/messages /var/log/secure /var/log/cron /var/log/yum.log
HTCondor	/var/log/condor/
ARC-CE	/var/log/arc/ /var/log/arc/bdii/
ARGUS	/var/log/argus/pap/ /var/log/argus/pdp/ /var/log/argus/pepd/
Squid	/var/log/squid/
Apache	/var/log/httpd/
Ansible	/var/log/ansible/ /var/log/alerts/

Loki does not index the content of the log files, but rather attaches a set of labels to each log stream to enable queries. Grafana is then used to search and display these logs (see Section-5 for further details). Logs are kept for a period of six months, which is twice the mandatory requirement for a WLCG Tier-2 site. During this time, they are transferred from the live search table, which goes back one month, to disk storage.

4 Alerting and Automation

Alerts are generated according to the outcome of conditional tests that involve matching particular metrics or queries against certain criteria. Both Prometheus and Loki can be configured to trigger alerts when a custom set of conditions is met. Prometheus implements such an option by default [10], while Loki uses an internal component called Ruler [20] (available since Loki v2.0) for this purpose.

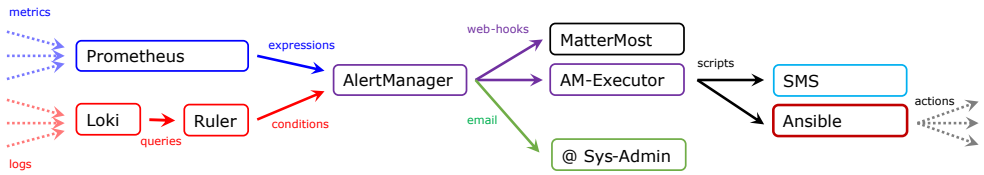

Figure 3. Chain of interacting systems for central monitoring and automated recovery. Each block represents a service.

4.1 AlertManager

To handle alerts, we use the default plug-in AlertManager [15], a service that collects alerts and takes care of de-duplicating and grouping them, before routing them to the correct receiver. These alerts, in turn, can send automated messages or trigger custom actions. The handling of alert messages is defined in the AlertManager configuration, where a set of routing rules match incoming alerts with a list of receivers (e-mail addresses or web-hooks) for

distribution, according to the type and severity of each alert. As anticipated in Section-1, we can achieve some level of automation by encoding simple recovery actions related to known issues into a set of Ansible playbooks that are executed when a specific alert is triggered.

Figure-3 summarises the interacting systems that are involved in the automation chain. Monitoring and logging were described in previous sections, and further details on automated recovery will be given in the following sections.

4.2 AM Executor

The Prometheus AM-Executor [16] is an HTTP server that listens for alert messages and executes a given set of commands with alert details passed as environment variables. Alert messages are sent by AlertManager as web-hooks to TCP port 8080. Custom commands matching a specific alert are defined in the AM-Executor configuration. Such a tool can be installed on the server that runs AlertManager, on worker-nodes, or elsewhere, such as on a central management server. After testing different set-ups, we conclude that the simplest, most efficient configuration is to install AM-Executor on the management server that runs Ansible. This solution gives access to the full power of Ansible remote management, where a variety of playbooks can resolve virtually all foreseeable issues.

4.3 Automated Recovery Actions

At present, the automated recovery system is still undergoing evaluation. However, we have identified several use-cases where automated actions can be safely performed.

Table 2. Alert conditions (expressed by metric names) and associated recovery actions. The label *node* refers to the affected machine, the label *all* means all worker-nodes.

Alert condition	Alert Name	Action	Mail Chat	SMS
avg(node_sensors_ambient{*all*}) > 25	TooHot	/	✓	✓
avg(node_sensors_ambient{*all*}) ≥ 30	WayTooHot	shutdown{*all*}	✓	
node_hwmon_temp{*node*} > 80 ∨ node_sensors_hddtemp{*node*} > 35	CpuHot HdHot	shutdown{*node*}	✓	
$\frac{\text{node\_filesystem\_free}\{node\}}{\text{node\_filesystem\_size}\{node\}} < 0.02$	InstanceFull	clean{*node*}	✓	
avg(node_condor_cpu{*node*})[1*h*] ≤ 1 ∧ avg(node_condor_cpu)[1*h*] > 10	NodeLazy	reboot{*node*} + enable{*node*}	✓	
rate({*node*} | ~ "[Ee]rror"[1*h*]) > 0.1	ManyErrors	disable{*node*}	✓	

Table-2 lists a few examples of conditions and consequent recovery actions that we have tested; most are based on Prometheus metrics, while one is based on logging information from Loki. Below is a more wordy description of them:

- **Global Temperature:** When the room temperature rises above 25°C, a warning message is sent by SMS to the person on duty. If the temperature keeps rising, at 30°C a critical alert triggers a complete shut-down of all worker-nodes in the affected room.
- **Local Temperature:** If either the CPU or hard-disk temperature of any node rises above certain thresholds (80°C for CPU and 35°C for HD), the affected machine is powered off.
- **Disk Full:** If the file-system of a worker-node gets too full (above 98% usage), a clean-up of the CVMFS cache is triggered.

- **HTCondor jobs:** If a node does not run any job for more than an hour despite available work, the node is rebooted and HTCondor is re-enabled.

- **Excessive Errors:** If a worker-node has too many error messages in its logs (over 30 errors within a one-hour period), HTCondor is disabled and details are sent by e-mail to a sys-admin for further investigation.

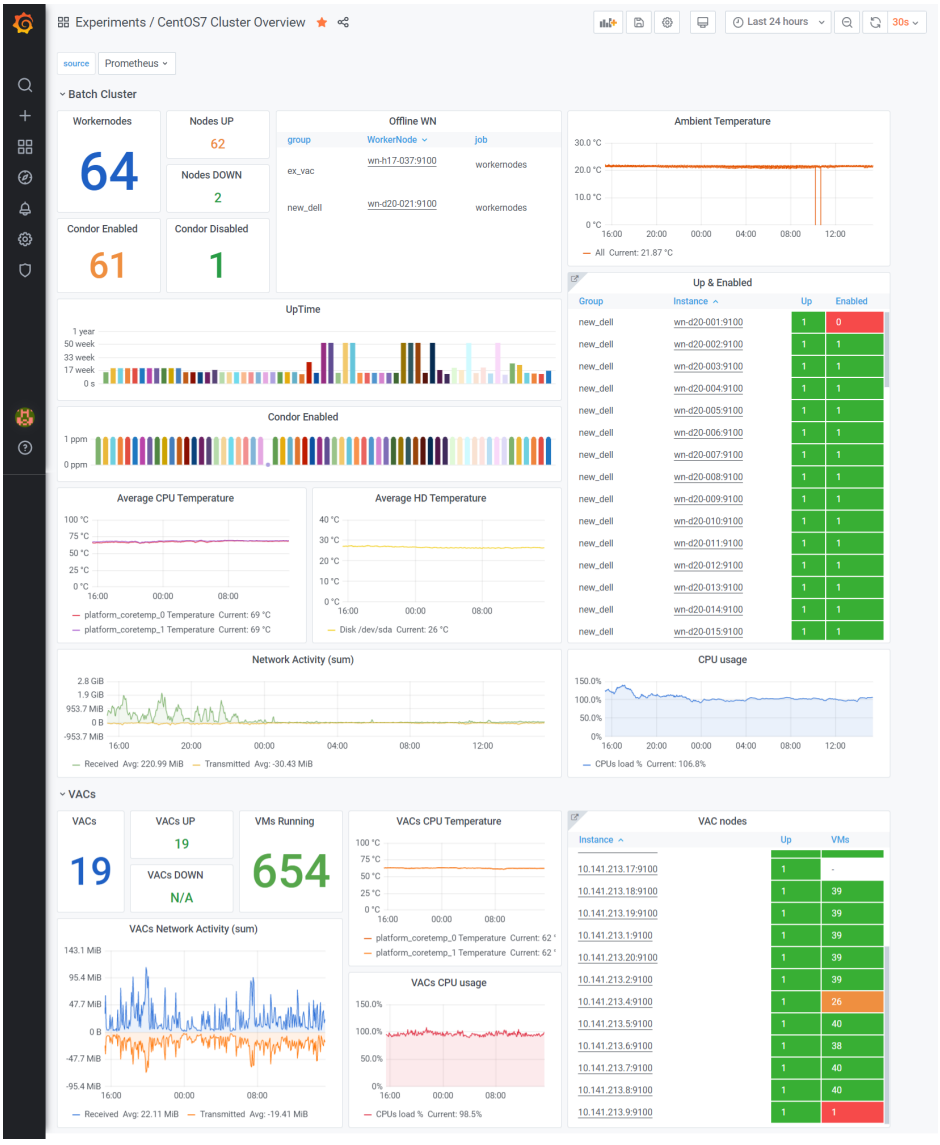

Figure 4. Cluster Overview dashboard on Grafana. Note the up-time plot (left) and the list of worker-nodes and VAC machines alongside their corresponding status (right).

5 Visualization & Dashboards

Grafana [12] is an analytic platform that provides the interface and graphical tools to query, visualize and explore metrics and logs. It reads data from any compatible data-source and displays them in customisable dashboards. As a monitoring tool, it allows sys-admins to easily identify issues and to dig into service logs for further investigation. We have built a number of dashboards to fit our monitoring needs, including single node metrics and collective indicators:

- **Cluster Overview** displays the general status of the cluster: the number (and a full list) of active nodes, average temperatures, global resource utilization (CPUs and RAM), network traffic and up-time (see Figure-4).
- **Batch System Monitor** shows the status of batch jobs (running, idle and queued), the run-time of current jobs, load on each CE, and usage statistics broken down by VO.
- **Single Nodes Monitor** displays node metrics such as temperatures (ambient, disk and CPU), current load, memory usage, file-system size, network statistics, and a count of running jobs per VO. Similar dashboards, incorporating small variations on this general design, are used to monitor essential services.

Queries are formulated within the Grafana interface using the Prometheus Query Language (PromQL [21]). This is a powerful tool for exploring and interrogating the collected logs: not only does it let us identify particular messages of interest (for example, those containing a certain text string or which originate from a specific machine within a particular time-frame) but it lets us exploit more complex queries incorporating calculations and distributions to generate real-time graphs of cluster behaviour and activity.

As an automation tool, Grafana provides the means to quickly check if and when a recovery action has been taken, by reporting the state of affected machines (such as up-time), and by providing ready access to the relevant logs and Ansible output. Furthermore, each alert generates an automated e-mail and also a message to our MatterMost [22] instant messaging system, so that no action is ever taken without the concerned sys-admins being notified.

6 Conclusions and Outlook

The use-cases that we have identified so far can be easily handled by hard-coded rules based on specific conditions and well-known solutions. Additionally, having converged all the available cluster intelligence to a single location, it is now feasible to implement more complex conditions and more articulated solutions. For example, we are now exploring the automated re-provisioning and re-installation of faulty worker-nodes or even services.

A sys-admin will need to configure each service only once. When a working configuration has been achieved and properly encoded in Ansible, any issue that arises – either a hardware failure or a broken software – can be resolved by restoring the initial working configuration. The small overhead needed to encode the configuration steps into an Ansible playbook immediately becomes worthwhile when the playbook is used a second time. The re-initialization of a service can be performed manually by a sys-admin; in full automation by a recovery system; or by using a clever combination of both, depending on the frequency of any given issue.

The monitoring system will be extended to incorporate networking by collecting sFlow and SNMP information from routers and switches. Some attempt has already been made using the ElasticSearch framework [23], but the integration of that with Grafana turned out to be tricky, and instead we have decided to look for a cleaner solution.

In the future, following the general guidelines given by CERN Operational Intelligence [24], the system could be extended to encompass a larger set of issues by exploiting anomaly detection algorithms and other Machine Learning tools to identify recurring patterns and irregularities in the cluster's work-flow.

References

[1] G.Roy et al., J.Phys.Conf.Ser. **513**, 062040 (2014)
[2] A.McNab et al., J.Phys.Conf.Ser. **513**, 032065 (2014)
[3] S.Weil et al., Proc. **OSDI'06** (2006)
[4] H.Thostrup et al., Proc. **ECG'05**, 861-871 (2005), doi:10.1007/11508380_88
[5] ARGUS, https://wlcg-ops.web.cern.ch/argus (2021)
[6] T.Tannenbaum et al., The MIT Press (2002), ISBN:0-262-69274-0
[7] CVMFS, https://cernvm.cern.ch/fs/ (2021)
[8] Ansible, https://www.ansible.com/ (2021)
[9] GitLab, https://about.gitlab.com/ (2021)
[10] Prometheus, https://prometheus.io/ (2021)
[11] Loki, https://grafana.com/oss/loki/ (2021)
[12] Grafana, https://grafana.com/ (2021)
[13] node_exporter, https://github.com/prometheus/node_exporter (2021)
[14] PromTail, https://grafana.com/docs/loki/latest/clients/promtail/ (2021)
[15] AlertManager, https://github.com/prometheus/alertmanager (2021)
[16] AM-Executor, https://github.com/imgix/prometheus-am-executor (2021)
[17] VictoriaMetrics, https://victoriametrics.com/ (2021)
[18] S.M.A.R.T., https://www.linuxjournal.com/article/6983 (2004)
[19] IPMI tool, https://github.com/ipmitool/ipmitool (2021)
[20] Loki's Ruler, https://grafana.com/docs/loki/latest/alerting/ (2021)
[21] PromQL, https://prometheus.io/docs/prometheus/latest/querying/basics/ (2021)
[22] MatterMost, https://mattermost.com/ (2021)
[23] ElasticSearch, https://www.elastic.co/elastic-stack (2021)
[24] CERN Operational Intelligence, https://operational-intelligence.web.cern.ch/ (2021)