DISTRIBUTED MONITORING SYSTEM BASED ON ICINGA

C. Haen*, E. Bonaccorsi, N. Neufeld, CERN, Geneva, Switzerland

Abstract

The LHCb online system relies on a large and heterogeneous IT infrastructure: it comprises more than 2000 servers and embedded systems and more than 200 network devices. Many of these equipments are critical in order to run the experiment, and it is important to have a monitoring solution performant enough so that the experts can diagnose and act quickly. While our previous system was based on a central Nagios server, our current system uses a distributed Icinga infrastructure. The LHCb installation schema will be presented here, as well some performance comparisons and custom tools.

INTRODUCTION

LHCb [1] is one of the four large experiments at the Large Hadron Collider at CERN. The infrastructure of networks and servers deployed in order to manage the data produced by LHCb and control the experiment are critical for its success. While for the control and monitoring of detectors, PLCs, and readout boards an industry standard SCADA system PVSSII has been put in production, at a lower level the network infrastructure and resources used by each server in the LHCb Cluster need to be monitored. This is because the PVSSII in the ECS (Experiment Control System) depends already on a lot of systems such as the shared storage, the network, DNS (Domain name Server) and others. In monitoring terminology, one uses 'host' to refer to a machine such as a server, and 'service' to refer to all kind of software, application or resource like CPU or disk space, etc. A 'service' is then applied to an 'host'. Over time, the amount of services and hosts to be monitored increased up to the point where the previous monitoring system shown on Fig. 1 based on Nagios [2] did not perform sufficiently, and we decided to search for a more scalable solution.

INSUFFICIENT PERFORMANCES

Nagios monitors hosts and services with tiny single-purpose programs called plugins that are executed periodically in order to get the status of a monitored entity [3] (host or service). Basically, executing a plugin consists in forking, creating and closing a socket. Thus, monitoring 40 000 entities induces a huge load on the central server running Nagios [4]. The result is a big latency in the checks and then in the alerts received by the experts.

* christophe.haen@cern.ch

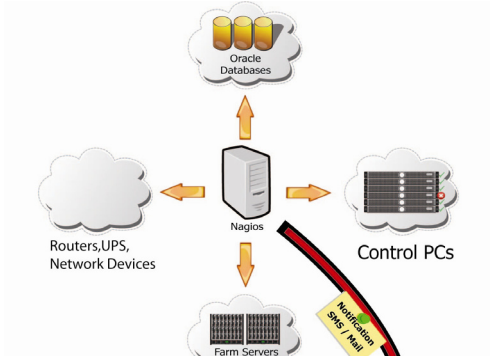

Figure 1: The previous Nagios installation.

NEW INFRASTRUCTURE

In order to improve our system, two changes took place: replacing Nagios by Icinga [5]; and going from a central to a distributed architecture.

Icinga is a fork of Nagios. There are several reasons to motivate the choice to switch:

- The support of the Icinga community is better.
- Extra features [6]: among them, the possibility of using a database as backend which can be queried with an API. This is particularly interesting regarding some other projects on going at LHCb.
- Nagios configuration files are compatible with Icinga.

Going for a distributed system is an obvious solution to avoid the bottleneck presented in the previous section. Three different possibilities to distribute our monitoring have been explored.

Independent Icinga Instances

The first option is to have several independent Icinga instances running, each of them monitoring a specific group of entities. The major drawback of that solution is the difficulty to have a redundancy in the checks: having the same checks executed by different instances would mean having all the alarms several times, since there is no information sharing between the instances.

Central Instance with Distributed Instances

In this setup, several instances would monitor specific entities, but they would all report to a central instance, whose goals are only to gather the check results, display it to the users, and trigger the notifications. This setup is possible with Icinga core [7]. If one of the distributed instance is dead, the central server will actively do the test,

thus ensuring redundancy for the distributed nodes. This solution has been seriously considered, but the configuration becomes complicated when you have many entities to monitor: the central server has to know all the entities, but each distributed server has to know only its own. In order to balance the drawback of this setup we developed a few tools to ensure consistency and dependencies:

- Oracle database backend: the database schema can be used to store a complete configuration for Icinga.
- Web frontend: based on symfony [8], this tool provides a global overview of the configuration, and allows non experts to use the monitoring services, by defining their own entities.
- Generation tool: thanks to a set of perl scripts, the configuration is extracted from the database, written in files readable by Icinga, and properly shared between the Icinga instances.

Even by using these tools, the management of large amount of entities was difficult: the frontend performances were not good enough to cope with our large installation. For this reason, we decided to give up that solution.

Central Icinga Instance with Distributed Workers

The third option is to have a single instance of Icinga which would schedule the checks and deal with the results, but having other servers (called 'workers') actually perform the checks. This is made possible by mod_gearman [9], an Icinga/Nagios broker module. The load induced on the worker is negligible, whereas the central server is fully busy. Mod_gearman is a module based on Gearman [10], a generic framework to distribute applications. It uses a client/server model as shown in Fig. 2. The server manages queues that clients use to get their tasks and give their results.

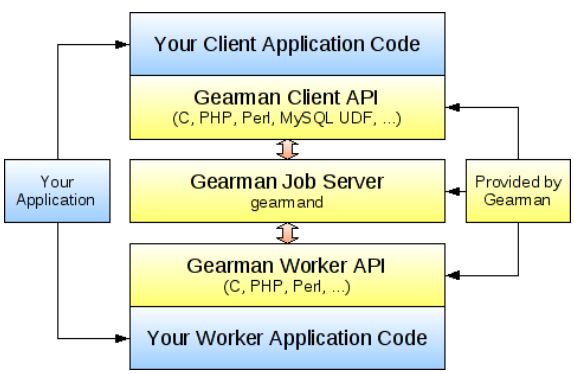

Figure 2: Gearman Stack.

Mod_gearman intercepts the checks triggered by Icinga, and puts them into queues managed by Gearman. The client side is a light weight program which simply gets the path of the executable, runs it, and put the result back in a result queue on the server side. Note that the executables need to be present on all workers. The elements in the result queue are then passed to Icinga, which processes them as if it had executed the checks itself.

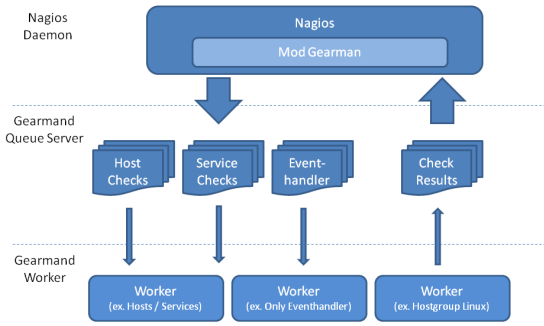

Figure 3: Mod gearman.

By default, all the checks are put in the same default queues showed in Fig. 3. This can be a problem if some workers cannot access the machine they should check, for network security reasons for example. To solve this issue, mod_gearman offers to put checks in separate custom queues, according to the group they belong to. These groups are groups of hosts or services defined in the Icinga configuration files: one can define one group for each separate network, and configure workers to fetch checks only from appropriate queue.

This is the solution we decided to implement in LHCb, and coupled it with a configuration schema described in the next section.

The other interesting feature proposed by Icinga is the database backend. With the idomod and ido2db modules [11], Icinga will log the result of every action and check-result in a database. The information can then be queried using a PHP API. This is a remarkable tool to use the Icinga results as input for other projects. Unfortunately, we found a deadlock in ido2db, for which a ticket has been open, which make it unusable for the moment.

Finally, to make sure we can always access our monitoring services, the server running Icinga is connected to the LHCb network and to the CERN network. Thus, even if a big network problem occurs in LHCb, the monitoring information can still be accessed through independent networks.

NEW SCHEMA

The whole LHCb infrastructure is made up of thousands of machines. We can divide in groups which are doing the same tasks, and thus must be monitored the same way.

Rewriting all the time the same complete configuration file for the different machines would be extremely tedious and error-prone. Fortunately, one can use two features offered by Icinga to avoid that:

- Hostgroup: an hostgroup simply defines a group of hosts. A host can be in several hostgroups, and a hostgroup can be member of another hostgroup.
- Inheritance: a host can inherit the configuration of another host, in the manner of the object oriented programming paradigm. The inheritance will only carry over the host monitoring options, not the services applied on the parent host. For example, the child will inherit the check frequency from its parent, but the service checks that are applied to the parent will not be applied to the child.

Thus by defining precisely the roles of all the machines, one can come up with a complete logical tree like in Fig. 4, which is then implemented as two separate but homeomorph trees. The first one is made of hostgroups: each node is member of the parent node. The service checks are always applied to a hostgroup in this tree, and never to a single host. The second tree is implemented with hosts, and each node inherits from its parent node: each level brings more specialized options, and the leafs of the tree are the machines themselves.

- Adding a new host consists in defining a name and an address, assigning it to the right hostgroup and inheriting from the right template (both being at the same in the logical tree). It will then be monitored exactly the same way as all the machines having the same role.
- Changing the way a functional group of machine is monitored is done by changing a single configuration file: for example, one can add a new service check on all of the file servers by adding this check only once to the hostgroup containing all the file servers.

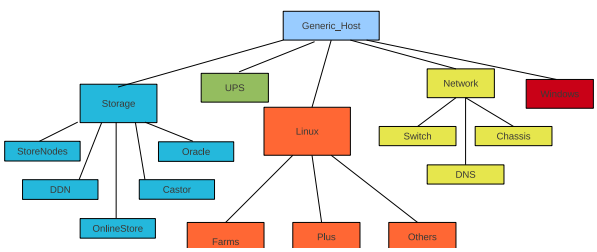

Figure 4: Part of the tree used to define configuration.

NEW PERFORMANCES

The new installation shown in Fig. 5 is now running 36000 service checks on 2100 hosts, with 50 gearman workers, and the performances are now on average 2760% better, as shown in Tables 1 and 2:

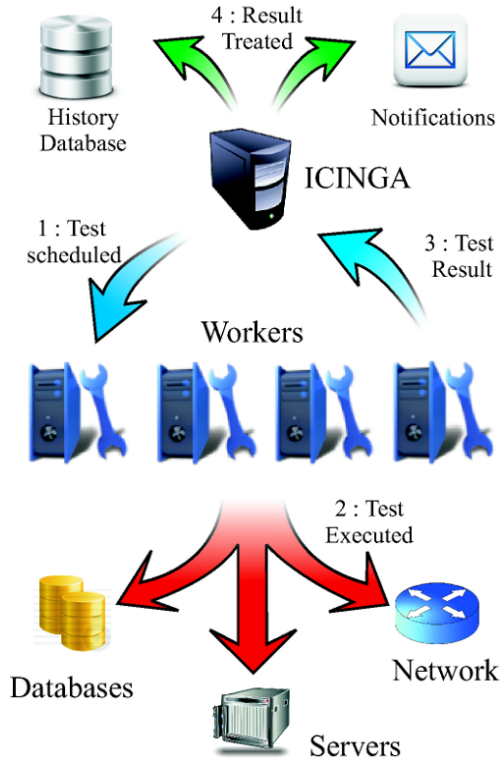

Figure 5: The new LHCb Icinga infrastructure.

Table 1: Single Nagios Instance Performances

	Min.	Max.	Average
Service checks latency	320 sec	578 sec	328 sec
Host checks latency	0 sec	401 sec	318 sec

Table 2: Central Icinga Instance and Distributed Workers Performances

	Min.	Max.	Average
Service checks latency	0.03 sec	57 sec	14 sec
Host checks latency	0 sec	35 sec	12 sec

The latency represents the difference of time between the scheduled execution time of the check, and the actual execution time.

CONCLUSION

Icinga has been running for two months in parallel with Nagios. The notifications have always been quicker and more up to date. We retired Nagios a month before this paper has been written.
In addition to that, the way our workers are spread over

the LHCb network, and the way the central Icinga server is connected to the external network ensures to have in almost any case access to our monitoring information.
Nevertheless, the central Icinga instance is still a single point of failure. A solution for this could be to set up a cluster of identical machines running in active-passive mode: if the server running the central Icinga instance crashes, a fail-over will happen and another server will take over.

REFERENCES

[1] A. Augusto Alves et al. The LHCb Detector at the LHC. JINST, 3:S08005, 2008.

[2] http://www.nagios.org/about

[3] W. Barth "Nagios, Systems and Network Monitoring" (2006)

[4] E. Bonaccorsi, Niko Neufeld "Monitoring the LHCb experiment computing infrastructure with Nagios" TUP001 ICALEPCS 2009

[5] https://www.icinga.org/about/

[6] https://www.Icinga.org/Nagios/

[7] http://docs.Icinga.org/latest/en/distributed.html

[8] http://www.symfony-project.org/

[9] http://labs.consol.de/lang/de/Nagios/mod-gearman/

[10] http://gearman.org/

[11] http://docs.Icinga.org/1.3.0/en/db_components.html