LHCb ONLINE LOG ANALYSIS AND MAINTENANCE SYSTEM*

J.C. Garnier[†], L. Brarda, N. Neufeld, F. Nikolaidis, CERN, Geneva, Switzerland

Abstract

History has shown, many times computer logs are the only information an administrator may have for an incident, which could be caused either by a malfunction or an attack. Due to the huge amount of logs that are produced from large-scale IT infrastructures, such as LHCb Online, critical information may be overlooked or simply be drowned in a sea of other messages. This clearly demonstrates the need for an automatic system for long-term maintenance and real time analysis of the logs. We have constructed a low cost, fault tolerant centralized logging system which is able to do in-depth analysis and cross-correlation of every log. This system is capable of handling O(10000) different log sources and numerous formats, while trying to keep the overhead as low as possible. It provides log gathering and management, Offline analysis and online analysis. We call Offline analysis the procedure of analyzing old logs for critical information, while Online analysis refer to the procedure of early alerting and reacting. The system is extensible and cooperates well with other applications such as Intrusion Detection / Prevention Systems. This paper presents the LHCb Online topology, problems we had to overcome and our solutions. Special emphasis is given to log analysis and how we use it for monitoring and how we can have uninterrupted access to the logs. We provide performance plots, code modification in well-known log tools and our experience from trying various storage strategies.

INTRODUCTION

LHCb [1] is one of the four large experiments at CERN which takes data from proton-proton collisions at the LHC. The control of the experiment and the data acquisition from the detector electronic channels to the permanent storage is managed within the Online Cluster. It is separated from the CERN IT infrastructure and managed entirely by LHCb people. The cluster is made of about 2000 Linux machines, 200 Windows machines and 60 switches and routers. Windows machines are the controllers of the detector systems. The Event-Filer farm [2], and all the other Data Acquisition (DAQ) servers are running under Linux. DAQ Software is controlled and monitored via the Farm Monitoring and Control System (FMC [3]). We use a uniform control interface for the detector and for DAQ software: the SCADA system PVSS [4].

The readout boards [5] are the interface between the detector links and the network links. An embedded computer allows the user to access and configure the entire board. It is running the Scientific Linux at CERN distribution [6].

Most of our machines are equipped with Baseboard Management Controllers (BMC) and accepts the Intelligent Platform Management Interface (IPMI [7]).

All these systems - Operating Systems, PVSS, DAQ software, IPMI, etc. - are generating numerous logs. They are traditionally consolidated locally on the system producing them. There are some limitations:

- Investigating problems can lead to correlating numerous log files from numerous locations.
- Once a machine is down, the local logs are unreachable and the problem investigation can only starts once the machine is up again.
- In case of hard drive failure, logs are lost.

We therefore implemented a central logging system, highly reliable and available, which collects logs from every system, consolidates them hierarchically and distributes them so that they can be reachable from numerous locations. Logs are then processed for indexation and analysis.

The first section presents the study and the deployment of the central logging system. The second presents the purpose of log analysis at LHCb, the tool we chose and its configuration. The third presents the status and the performance of the system, its limitations and its future.

CENTRAL LOGGING

Linux traditionally uses the *syslog* daemon [8] to manage logging from the applications and from the system. Logs are identified by a facility and a severity. The facility is usually used to define the source of the message. There are 20 facilities; hence several sources have to share the same facility in large systems. It is up to the system administrator to configure this correctly. The severity is an eight-level ladder used to classify the message importance. The daemon task is then to write different files for any combination of facility and priority. *Syslog* can be run as a client / server application. A node will send its messages to a remote node.

Syslog's features are sufficient to design a central logging system, but they were not flexible enough for us. Sharing 20 facilities amongst one node is acceptable, but not among 2200 nodes. Our system has to be able to differentiate a large number of systems, it should be able to write

* This research project has been supported by a Marie Curie Initial Training Network Fellowship of the European Community's Seventh Framework Programme under contract number (PITN-GA-2008-211801-ACEOLE).

† jean-christophe.garnier@cern.ch

several files per nodes. A few thousand files are expected. Figure 1 illustrates the file system architecture.

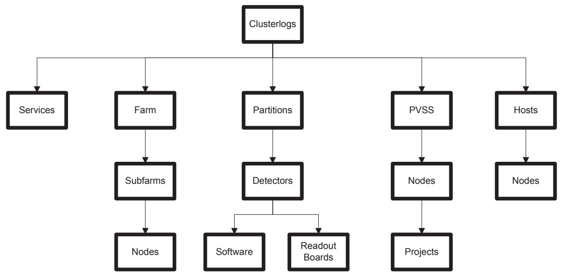


Figure 1: Log File System structure. The directory is divided first by themes. Then there are sub-categories corresponding to subsystems, then nodes, and then projects or software.

Rsyslog [9] is an enhanced *syslog* daemon. A few noticeable features are:

- A more complex analysis of the log message, using tags, regular expressions, sources, etc.
- The possibility to convert files to the *syslog* protocol, hence to get the logs from applications which are not using *syslog*.
- Multiple queues and multiple threads.
- Aggregate log lines into batches before to write them, to maximize IO performance.
- Message discard according to watermarks and log severity.

From these features, one can notice that *rsyslog* is oriented for setting up a centralized logging system. *Rsyslog* uses the *syslog* protocol so it is compatible with the *syslog* daemon. We deployed *rsyslog* on most of our cluster as client instances, except on embedded computers where it is preferable to run only *syslog*, which is much lighter.

The logging system is implemented with a few Linux servers and thousands of clients. It is described in figure 2. The log servers are configured as a cluster. The cluster nodes are identical commodity servers. Processing is not a critical issue here so they have only a single processor chip of four cores. They are equipped with 8 GB of memory. They have two SATA II hard drive disks of 2 terabytes configured as mirrors in *RAID 1*. Each node has two network interfaces. The first is used for control, for receiving the log messages and for exporting the log file system to log clients for analysis. The second is used for meta-data transfer and for cluster synchronization.

Corosync [10] manages the communication between the nodes of the cluster. On top of it, *pacemaker* [11] manages the applications that are critical for the cluster, like the various logging mechanism, virtual IP addresses, etc. It makes sure that the nodes and the services that they provide are running correctly, and distributes them amongst the cluster. Logs are written in a common area for all the cluster nodes, using *gluster* [12]. This is a file system that we use to replicate the information between all the nodes of the cluster, in order to increase the reliability of our system. Hence, if we have four machines with 2 terabytes hard disk space, our cluster area will only have a maximum of 2 terabytes space. *Rsyslog* is running as a server on every cluster nodes. This particular daemon is not managed with pacemaker as it has to run on every instance. It is simply managed via *chkconfig*. This is the main logging software, but there are a few more custom software. They are managed by pacemaker, and they process custom logging protocols developed for LHCb. The cluster is configured an Active/Active way. All nodes are processing logs coming from clients in parallel. In order to do that, we use a virtual IP address that every node recognizes as theirs. This IP address is bounded to a multicast Ethernet address that the switch uses to send data to all servers. Each server receives every log messages, but they will only process a subset of them, according to a hashing algorithm managed by *netfilter* [13, 14].

The log servers all share the same configuration for *rsyslog*. The current configuration is awfully complex, but it is interesting to stress that it currently does not rely on multi-queuing and multi-threading as we observed that performance were decreasing so much that no logs were written in the end. We suspect a deadlock or live-lock issue and further investigations are required before to report about that to the developers. The daemon is configured to write batches of several lines, in order to improve IO performance, trying to do not delay the log information too much for the user.

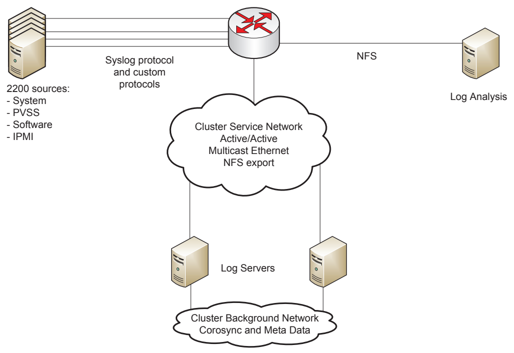


Figure 2: Architecture of the logging system. 2200 clients are logging information using the *syslog* protocol into an Active/Active cluster. This cluster exports the log files via NFS for log analysis. Log analysis is then performed by user applications and a host based intrusion detection system.

Clients are running both Windows and Linux. Under Linux, *rsyslog* sends the system logs, most application logs, PVSS and IPMI logs to the log cluster. Our farm and our data acquisition servers are however running applications which are not logging to *syslog* but to FIFOs which are then read by FMC loggers. These loggers can then directly write a file or forward logs to a remote FMC logger. The log cluster runs FMC loggers, but some of their proprieties prevent them to run Active/Active like *rsyslog*, so pacemaker makes sure that only one server runs one instance of *rsyslog* for a same system. The event logging system of Windows is slightly different and does not rely on *syslog*. The system relies there on *Snare* [15] and *Epilog* [16] to respectively send event log messages and PVSS logs to the cluster using the *syslog* protocol.

The log cluster exports the log file system via NFS to some servers, so that users can access all logs easily. A special server is performing log analysis.

LOG ANALYSIS

Log analysis consists in reading the log files to look for patterns or sequences of lines. Most of the lines will not be interesting and nothing will be produced, in some cases a message is written in a dedicated log file which is a summary of what was important. Really important log messages will trigger alarms, sent by e-mail or text message, and they could also trigger automatic responses to fix the problem.

Several tools exist and are open source, but our choice is the Open Source Host-based Intrusion Detection System OSSEC [17]. Performing log analysis using a Host-based Intrusion Detection System (HIDS) comes along with a wider strategy of security in LHCb. It will be peered with an IDS performing network analysis.

OSSEC brings numerous features as it is a client server HIDS, but our primary use here will be to configure it as a log analyzer. It consists in defining decoders, which are templates that a log line can match or not. If a line matches a decoder, then OSSEC can apply rules to this line, in order to study it, to define its degree of importance, and to decide if the administrator should be warned, immediately or later, and whether to run or not an automatic action to fix the problem.

The OSSEC configuration is complex for our cluster, and there is still a lot of tuning to be done. In order to keep tracks of old configurations, the OSSEC rules and decoders are written into a GIT [18] repository.

RESULTS

We successfully implemented a central logging service with log analysis. It collects most of the identified logs of the LHCb Online system and writes them in more than 12000 files. Figure 3 shows the performance of a node over one month. The network is not overloaded but some peaks can make performance quite weak. The scalability, the reliability and the high availability requirements are fulfilled. There are however a lot of improvements to implement.

The typical issue of such system is the "split brain" issue. It happens when one or more members of the cluster cannot communicate with the other members for a while, and are starting to work independently from each other. Pacemaker can make sure that such situation does not happen, whereas it occurred with gluster a few times. It is not a critical issue at all there, as once the cluster is unified, gluster will recover files which were written by both sub-clusters. Figure 4 shows the IO performance of the hard drive of a node when such recovery occurs. The nodes are basically unable to write any log messages for a while as their hard drive disks are monopolized by gluster to check the 12000 files of our system and to merge data from the sub-clusters. At first the log servers were designed with 4 GB of memory and the recovery was a critical moment as all log messages are cached in memory. Doubling the memory made this situation more comfortable. If the gluster area size increases much more, this recovery might take much more time and the caching might not be enough to overcome it. In that case, upgrading the hard drive disks to formats which can support a larger number of IOs would improve the situation.

Another issue comes from the Active/Active configuration of the cluster. Netfilter is a standard module of the kernel. Its project which manages the virtual IP address and the multicast Ethernet address is however written with a hard-coded debug logging. It prints a line for each packet received. The log files for the log servers are basically polluted with these messages. There are two solutions. The first is to use another netfilter module, more recent and not integrated into pacemaker yet. The second, that we adopted, is to patch the kernel module to remove the disturbing lines.

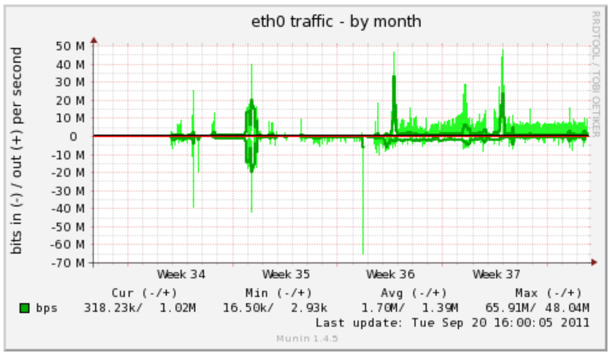
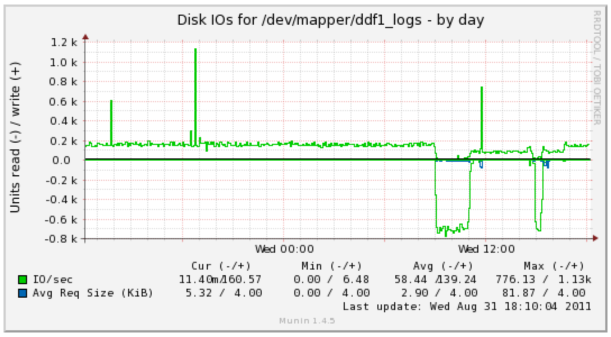


Figure 3: Log server throughput for one month. The input rate are log lines coming to *rsyslog* or custom loggers. The output rate is the NFS rate.

The system would ideally be real time, users could read logs remotely as soon as they were written locally by the nodes, and the log analysis would be performed at this moment as well. It is not possible yet for several reasons. In order to maximize IO performance, *rsyslog* is configured to write batches which produces a slight delay. Thereafter, our current system is unable to export the *gluster* file sys-



```
                      Cur (-/+)       Min (-/+)       Avg (-/+)        Max (-/+)
🟩 IO/sec             11.40m/160.57   0.00 /  6.48    58.44 /139.24   776.13 / 1.13k
🟦 Avg Req Size (KiB)  5.32 / 4.00    0.00 /  4.00     2.90 / 4.00     81.87 / 4.00
                                              Last update: Wed Aug 31 18:10:04 2011
```

Figure 4: Disk IOs on the RAID file system. The read peaks around Wednesday 12:00 were gluster split brain recovery situations. During these periods, all IOs operations are allocated for gluster and rsyslog cannot write most of the logs. It still performs caching.

tem via NFS, so it exports the real file system. Hence, the log analyzer and the users have to wait for the *gluster* cache to be flushed to disk. An alternative could be to user a *gluster* client instead of NFS.

LHCb, and more particularly Fotis Nikolaidis, contributed to the *rsyslog* module *imfile* which manages the translation of files to syslog. With this module, *rsyslog* polls files periodically, and logs the difference between the previous poll and the current poll. The new version of the module provide the option to use *inotify* [19] under Linux instead of polling. This tool allows applications to register in order to be informed when changes happens to files. Hence, changes in the log files are directly reported to *rsyslog*.

REFERENCES

[1] The LHCb Collaboration, Augusto Alves Jr and others, "The LHCb Detector at the LHC", JINST, 3:S08005, 2008.

[2] http://lhcb-trig.web.cern.ch/lhcb-trig/HLT

[3] https://lhcbweb.bo.infn.it/twiki/bin/view.cgi/LHCbBologna/FmcLinux#Introduction

[4] http://lhcb-online.web.cern.ch/lhcb-online/ecs/PVSSIntro.htm

[5] http://lphe.epfl.ch/tell1/

[6] http://linux.web.cern.ch/linux/

[7] http://www.intel.com/design/servers/ipmi/

[8] http://datatracker.ietf.org/wg/syslog/charter/

[9] http://www.rsyslog.com/

[10] http://www.corosync.org

[11] http://www.linux-ha.org/wiki/Pacemaker

[12] http://www.gluster.org/

[13] http://www.netfilter.org/

[14] http://security.maruhn.com/iptables-tutorial/x8906.html

[15] http://www.intersectalliance.com/projects/SnareWindows/index.html

[16] http://www.intersectalliance.com/projects/EpilogWindows/index.html

[17] http://www.ossec.net/

[18] http://git-scm.com/

[19] http://lwn.net/Articles/104343/