Archival, anonymization and presentation of HTCondor logs with GlideinMonitor

Marco Mambelli[1,1], *Mirica* Yancey[2], and *Thomas* Hein[3]

[1]Fermilab, MS120, PO Box 500, Batavia, IL (USA)
[2]Valparaiso University, Valparaiso, IN (USA)
[3]University of Illinois at Chicago, Chicago, IL (USA)

Abstract. GlideinWMS is a pilot framework to provide uniform and reliable HTCondor clusters using heterogeneous resources. The Glideins are pilot jobs that are sent to the selected nodes, test them, set them up as desired by the user jobs, and ultimately start an HTCondor schedd to join an elastic pool. These Glideins collect information that is very useful to evaluate the health and efficiency of the worker nodes and invaluable to troubleshoot when something goes wrong. This data, including local stats, the results of all the tests, and the HTCondor log files, is packed and sent to the GlideinWMS Factory. To access this information, developers and troubleshooters must exchange emails with Factory operators and dig manually into files. Furthermore, these files contain also information like email and IP addresses, and user IDs, that we want to protect and limit access to. GlideinMonitor is a Web application to make these logs more accessible and useful: it organizes the logs in an efficient compressed archive; it allows to search, unpack, and inspect them, all in a convenient and secure Web interface; via plugins like the log anonymizer, it can redact protected information preserving the parts useful for troubleshooting.

1 Introduction

The primary objective of this paper is to describe the GlideinMonitor system and to show its utility in a Glidein-based distributed High Throughput Computing (dHTC) system.

In this work, we first provide some background information about the GlideinWMS [1] system and the information it collects; secondly, we describe GlideinMonitor [2] and explain how it simplifies the activities of software developers and GlideinWMS operators; and finally, we show how the system can comply with restrictive privacy policies and still allow the work of troubleshooters and developers.

GlideinWMS is a Glidein-based Workload Management System leveraging HTCondor. It is an overlay system: at the upper layer it provides to the users reliable and uniform virtual clusters, green in Fig. 1. These are HTCondor pools with Machines matching what the users need. Different pools serve different user communities, aka VOs (Virtual Organizations). Underneath, GlideinWMS submits Glideins to heterogeneous resources

[1] Corresponding author: marcom@fnal.gov

leveraging again HTCondor, this time for scheduling and job control. The Glideins, aka pilot jobs, are regular jobs for the compute resource they are submitted to but are properly configured to become compute nodes for the virtual HTCondor cluster after starting an HTCondor startd daemon. The remaining components of the system are the Factory and the Frontend. The Factory knows how to submit Glideins to the resources: it supports many different resource types, from clusters to Grid sites, to commercial Clouds, it has a list of trusted and tested sites, and it knows what to expect from each one and the correct parameters to use when submitting Glideins. The Frontend monitors the user requests, selects the best resources to provide the virtual cluster for the users and requests the Factory to submit Glideins to those resources.

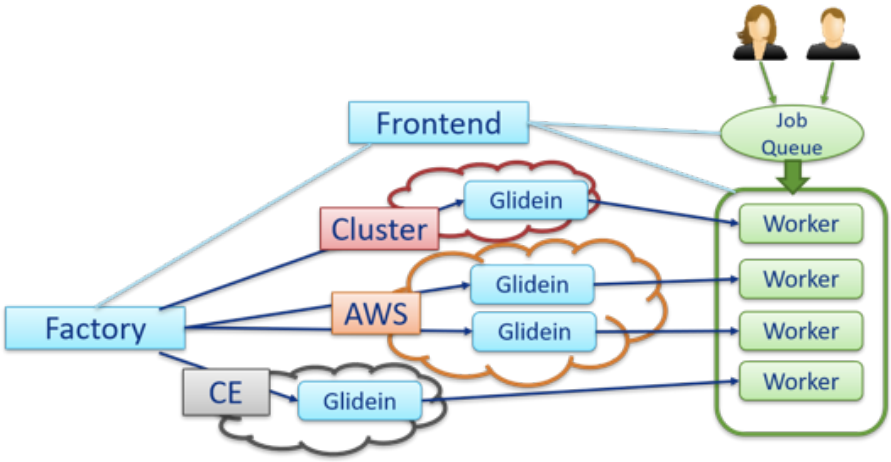

Fig. 1. GlideinWMS overlay system. The virtual cluster as seen by the users is in green, the GlideinWMS components and actions are in blue. The clouds represent examples of different computing resources and their gatekeepers: a local cluster, a commercial cloud like Amazon Web Services, and a Grid site with its Compute Element (CE).

The system is transparent to the users. It isolates them from the handling and the management of the sites: testing and adding new sites or new types of sites, or sites being unavailable. It protects them from many host failures: the Glidein can run multiple tests to validate a host and discard it if unreliable or unfit for the user jobs. The same Glidein can be reused for multiple jobs and run them one after the other or in parallel. Any Glidein failure before starting the user job is not affecting the users. These failures, together with the statistics about the nodes and the results of all the tests are used to evaluate the health of the sites and improve the system.

Each Glidein collects the job environment, measurements of the resources received, the results of standard and user requested tests, the customized configuration and logs from HTCondor, and statistics about the user jobs executed. The Glidein is compressing all this information and packing it in a JSON structured log and in its standard output and standard error streams. At the end of the Glidein's execution, HTCondor and the computing resource transfer the standard output and standard error back to the Factory that submitted the Glidein. A recent extension allows Glideins to upload log files also to monitoring servers, Web servers designated by the Factory or Frontend, accepting authenticated uploads. These uploads happen also during the execution of the Glidein, not only at the end, allowing more timely incremental updates.

The information collected can then be used by Factory operators to tune the configuration or validate new sites, by Site system administrators to fix problems with their resources, by VO operators to tune the Frontend configuration or to resolve authentication issues, and by developers to fix bugs and optimize the code. But there are a few difficulties with the use of the logs.

Out of all the people above, only Factory operators have access to the Glidein logs. Everyone else must request from them a copy of the desired files, usually by exchanging emails. And in normal deployments where multiple Frontends talk to multiple Factories, finding the right log files can become a lengthy process. Log distribution improved with monitoring servers, but it is still limited.

Furthermore, the Glideins' log files are in a packed format. The Factory installation contains tools to process those log files, e.g., to extract the HTCondor log files. So, anyone inspecting the logs must install these tools.

Last but not least, many sections of these files contain protected information about the jobs or the submitters that is preferable not to share, like user IDs, email addresses or IP addresses. Sometimes we may even be required by law to do so. All these are obstacles that limit or delay the access to information and result in additional work and longer times to troubleshoot and solve problems.

The Glidein monitoring infrastructure, GlideinMonitor and the Anonymization filter aim to eliminate these obstacles.

2 GlideinMonitor

GlideinMonitor provides a way for GlideinWMS operations teams to fetch, retain and distribute Glidein log files and allows users to easily search and inspect them. GlideinMonitor has two components as shown in Figure 2: a log files indexing system, the "Indexer", and a Web server which provides an interface to search, retrieve and inspect Glidein logs. Each component can be installed on the same host, or on a separate host, or as a microservice, as long as all can share a tags database and access the same file system where the log archives are stored.

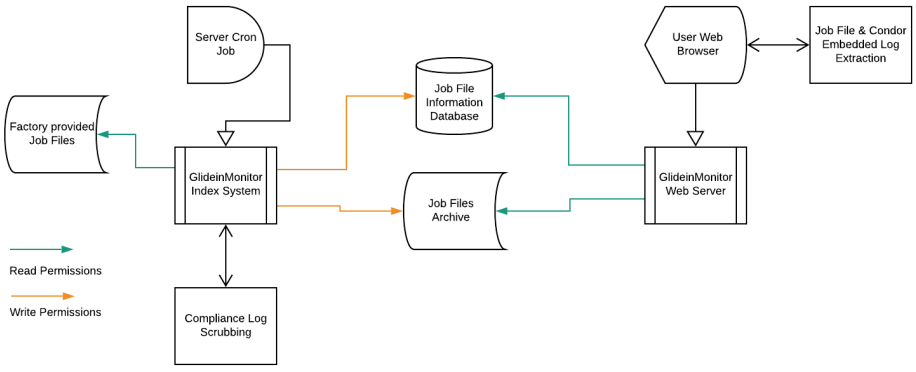

Fig. 2. GlideinMonitor architecture.

2.1 Indexer

The Indexer handles the Glidein log files provided by one or more GlideinWMS Factories or monitoring servers: it extracts indexing information, filters their content as desired, generates compressed archives, and updates the tags database with logs metadata. The indexer runs periodically to capture new Glidein logs and update old ones.

Unique Glidein and log file identifiers allow the indexer to group and update all log files from the same Glidein, also when they are provided by different servers.

The log files are provided to the indexer in its intake area, e.g., by copying them via rsync from Factories or monitoring servers. If they reside on the same host, GlideinMonitor could directly use the Factory log directory, but the copies allow for different retention policies in the different services.

The Indexer extracts key data from each log file and uses it to identify if it is a new file or an update, and also to populate the tags database. Then it bundles together and compresses all the log files from the same Glidein and adds them in a structured archive. Each archive is a hierarchical file system structure where logs of Glideins requested by different users or submitted by different Factories are separated to ease access control. Each file in the archive contains one version of all the log files from one Glidein, bundled and compressed.

Finally, the Indexer manages the filters. Any executable that given one version of the Glidein logs produces a new version can be a filter. A reference implementation is provided in the GlideinMonitor distribution, and it is easy to add new ones. The filter API specifies two options: filters that operate directly on the packed log files (Glidein stdout, stderr and structured log), and filters that operate on the unpacked logs and let the Indexer do the unpacking of the logs and the repacking with the filtered components. The Indexer also orchestrates the filters. When there are many filters, the Indexer determines which filters run in parallel, which are daisy chained, and which versions of the filtered log files to save in a new archive.

2.2 Web server

The Web server is an authenticated Web dashboard for users to interact with. The dashboard can serve multiple versions of the log archive, e.g., raw and filtered log files, each version only to the users that are authorized to view it. Once an archive is selected, the dashboard allows it to be searched for jobs in a table-based format, using the tags database. Specific restrictions, such as providing a date-time range and selecting which computing resources the user is interested in, can be added to limit the set of jobs displayed.

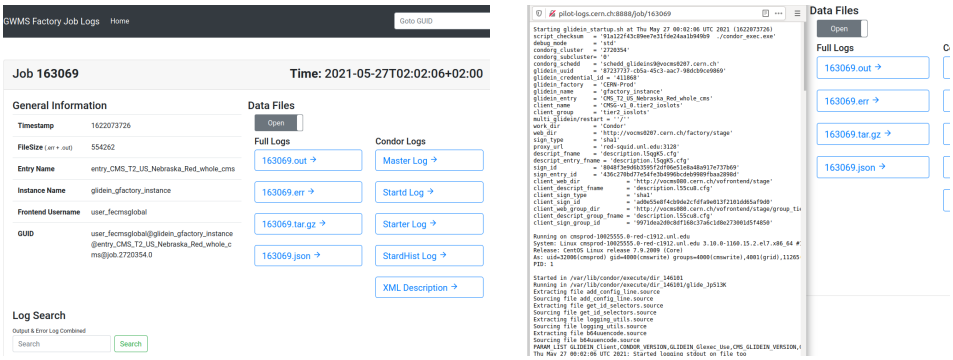

Fig. 3. Log views in the Web Server (3a, left, job view. 3b, right, extracted log)

Once a particular job has been selected, the Job view page, in Figure 3a, will provide general information about that Glidein such as the computing resource it ran on, the originating Factory, the creation time, and more. Links are also provided to either view or download all the HTCondor log files embedded in the Glidein stderr and the full Glidein logs from the archive. The Job view page uses client-side scripting to download the archived Glidein logs and unpack and serve them. The web browser extracts locally the Glidein log files, parses the extracted log files, and extracts and parses the hashed HTCondor logs within them as visible in Figure 3b. This improves responsiveness and reduces the load on the Web server, which only sends the archived job file as it sits on the disk. This allows also to take advantage of Web proxies close to the client to reduce the network load and reverse proxies to improve the server scalability.

The Web dashboard is powered by a RESTful API that other applications can use as well. A client can request a Glidein log list providing constraints like the ones in the dashboard page, it can request the job view information, or it can download a compressed log file bundle from the archive. Responses are in JSON, except when the raw log file bundles are requested.

3 Log Anonymization

Whenever storing information like in GlideinMonitor and even more when sharing it, it is important to follow the guidelines of Privacy-Preserving Data Publishing (PPDP), to respect the users' privacy and to comply with regulations like Europe's GDPR (General Data Protection Regulation). Therefore, we decided to develop and distribute a log anonymization plug-in.

The log anonymization plug-in is an irreversible filter based on k-Anonymity and uses regular expressions for the location of important user data. It is capable of locating user data such as IP addresses, usernames, full names, and other identifiers. It uses GlideinMonitor's standard plug-in API for filters with log expansion: it lets GlideinMonitor unpack the Glidein logs, and it filters all the resulting components, including HTCondor logs.

3.1 Research

Different factors can affect the design of an anonymization model: there are several privacy models, reversible vs. irreversible techniques, and different code models.

First, we compared reversible vs irreversible anonymization. Reversible anonymization allows for the recovery of data, often through a hashed lookup table. This ensures that it is capable of protecting data while preserving the original data use. Contrary to this, irreversible anonymization ensures the permanent loss of data which upholds the most data protection but less data use than reversible anonymization. Ultimately, the user and host identifiers that we plan to remove from the Glidein logs are of limited to no use for troubleshooting and support. Furthermore, we could keep an archive with more limited access with the original logs. So, we decided to use irreversible anonymization to ensure the most protection possible and to reduce the development effort.

Secondly, we examined several privacy models such as k-Anonymity [5, 6], l-Diversity [7] and t-Closeness [8]. k-Anonymity is basically the suppression or generalization of data by either omitting data or by increasing the range of data to obscure its exact value. Similarly, l-Diversity reduces the specificness of data but to a greater extent than k-Anonymity, by eliminating data, grouping similar categories of data or further generalizing data values and groups. t-Closeness is a more refined version of l-Diversity that focuses on the distance between two attributes and the "distribution of sensitive attributes within each quasi-identifier group" [3, 4]. Ultimately we decided on k-Anonymity for its simplicity of design and implementation. It is weak against background knowledge attacks but with a robust design we believe we can cover for that weakness easily.

Finally, we looked into two different possible code models for recognizing the data: Named Entity Recognition and regular expressions. Named Entity Recognition [9] allows the recognition of data by predefined categories and it is able to learn and be trained into further specifics. Regular expressions allow the recognition of data by patterns and are supported by most coding languages. Named Entity Recognition is more complex and regular expressions are sufficient to recognize the elements that we want to remove from the Glidein logs. They are easier to customize and will require less time to implement.

3.2 Implementation

The goal is to develop a filter that would automatically and accurately identify each user-identifying piece of information from a variety of file types, redact that information, and save the filtered log files for GlideinMonitor to archive and serve.

We analyzed and annotated a wide sample of different log file types from a variety of jobs to find the common denominators for when user data was revealed or referenced. We then began developing a regex script for IP addresses that found the pattern for IPv4 and IPv6 addresses and did an in-line replacement for each instance of the user's IP address.

This first version was performing an inline replacement cycling through all the lines of all the files. A scalability test revealed this solution as clunky, time-consuming and vulnerable to leaving partially anonymized logs. So, we switched to reading the whole file into memory, and using a faster regex-identifying method and bulk replace before writing the output. This solution is faster and removes the possibility of partially anonymized log files. The increased memory use is limited because the log files are relatively small.

Afterwards, we focused on identifying and removing general user information like email addresses and user IDs. The resulting new script has several methods to accommodate the

different patterns and log file types. Each method searches for a specific identifier, isolates the information on that line, and then strips the user information.

Finally, the script was added to GlideinMonitor as a plug-in filter and tested with the whole system. The filter was tested on a sample of Glidein logs from two VOs, including both successful and failed jobs, and the rules were general enough to anonymize all the files.

4 Conclusion

GlideinMonitor is a very useful tool to archive and share Glidein logs and, with its anonymization plug-in, it makes it possible to follow the guidelines of Privacy-Preserving Data Publishing and comply with regulations like GDPR. GlideinWMS provisions millions of Glideins every day, each running multiple jobs, serving many different VOs. It uses a handful of Factories and a dozen of Frontends submitting Glideins to a few hundreds of resources. This means that there are many different institutions and organizations involved, each operating part of the infrastructure. Glidein logs are useful to understand most problems, whether they are caused by the computing resources, the network, the software, the service configuration or user jobs. Normally a troubleshooter asks the Factory operators via email or via a ticket for the log files, an operator must search for them, and the troubleshooter receives, unpacks, and inspects them. To resolve a problem multiple requests may be involved, after troubleshooting actions, and each loop may take days if operators and troubleshooters are in different time zones. All this is simplified with GlideinMonitor and anonymized logs: the troubleshooter or developer can directly search and retrieve the log files without involving the Factory operators and GlideinMonitor is also preprocessing the files, giving direct access to the useful troubleshooting information. The result is a faster process involving less human labor.

GlideinMonitor can also be easily integrated with other tools. The latest prototype of the CMS monitoring infrastructure [10] is using GlideinMonitor to serve Glidein logs, linking it to a Grafana dashboard in CERN's MONIT.

Future work will involve a more precise classification and anonymization of the information in the log files, distinguishing Personal Information Identifiers (PII), Quasi Identifier (QI), Sensitive Attributes (SA), and Non-sensitive Attributes. This would allow more useful data without compromising or even improving privacy, e.g., preserving some IP addresses that disclose no user information and can be useful for troubleshooting. Another interesting development would be to analyze how robust is the current anonymization against linkage attacks, where the attacker combines the GlideinMonitor data with other sources or background knowledge to identify a user.

This work was done under the GlideinWMS project and the TARGET and SIST internship programs at Fermilab. This manuscript has been authored by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.

References

1. M. Mambelli, P. Mhashilkar, D. Box, I. Sfiligoi, D. Strain, et al, (2020) glideinWMS/glideinwms. Zenodo. http://doi.org/10.5281/zenodo.1309678, accessed: 2021-06-06
2. M. Mambelli, T. Hein, GlideinMonitor. United States. https://doi.org/10.2172/1605567 accessed: 2021-06-06

3. J. Vasa, P. Modi, *Review of Different Privacy Preserving Techniques in PPDP*, International Journal of Engineering Trends and Technology, IJETT, **59**, 5 (2018), https://arxiv.org/pdf/1808.04088.pdf, accessed: 2021-06-06
4. K. Rajendran, M. Jayabalan, M. E. Rana, *A Study on k-anonymity, l-diversity, and t-closeness Techniques focusing Medical Data*, International Journal of Computer Science and Network Security, IJCSNS, **17**, 12 (2017)
5. P. Samarati, L. Sweeney, *Protecting privacy when disclosing information: K-anonymity and its enforcement through generalization and suppression*. (2007) [Online] Available at: https://epic.org/privacy/reidentification/Samarati_Sweeney_paper.pdf, accessed: 2021-06-06
6. L. Sweeney, L. *Achieving k-Anonymity Privacy Protection Using Generalization And Suppression*, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, IJUFKS 10(5), pp. 571–588. (2002) doi:10.1142/s021848850200165x.
7. A. Machanavajjhala, D. Kifer, J. Gehrke and M. Venkitasubramaniam, *L -diversity: privacy beyond k-anonymity*, ACM Transactions on Knowledge Discovery from Data, 1(1). (2007) doi: 10.1145/1217299.1217302.
8. N. Li, T. Li and S. Venkatasubramanian, *Tcloseness: Privacy beyond k-anonymity and l-diversity*, ICDE 2007 IEEE 23rd International Conference on Data Engineering, (2007) doi: 10.1109/icde.2007.367856.
9. D. Bikel, R. Schwartz, R. Weischedel, *An Algorithm that Learns What's in a Name*. Machine Learning, 34 (1-3) pp. 211-231 (1999) https://link.springer.com/article/10.1023/A:1007558221122, accessed: 2021-06-06
10. F. Legger, V. Kuznetsov, C. Ariza Porras, C. Uzunoglu, R. Indra, *The evolution of the CMS monitoring infrastructure*, CHEP2021, to appear in the proceedings.