Analysis of data integrity and storage quality of a distributed storage system

Adrian Eduard Negru[1], *Latchezar* Betev[2], *Mihai* Carabaş[1], *Costin* Grigoraş[2], *Nicolae* Ţăpuş[1], and *Sergiu* Weisz[1]

[1]Faculty of Automatic Control and Computer Science, Politehnica University of Bucharest, Romania
[2]European Organisation for Nuclear Research (CERN), Geneva, Switzerland

Abstract. CERN uses the world's largest scientific computing grid, WLCG, for distributed data storage and processing. Monitoring of the CPU and storage resources is an important and essential element to detect operational issues in its systems, for example in the storage elements, and to ensure their proper and efficient function. The processing of experiment data depends strongly on the data access quality, as well as its integrity and both of these key parameters must be assured for the data lifetime. Given the substantial amount of data, O(200 PB), already collected by ALICE and kept at various storage elements around the globe, scanning every single data chunk would be a very expensive process, both in terms of computing resources usage and in terms of execution time. In this paper, we describe a distributed file crawler that addresses these natural limits by periodically extracting and analyzing statistically significant samples of files from storage elements, evaluates the results and is integrated with the existing monitoring solution, MonALISA.

1 Introduction

ALICE [1] stands for "A Large Ion Collider Experiment" and it is one of the 4 large experiments at the Large Hadron Collider (LHC) in the European Organization for Nuclear Research (CERN). To meet the processing and storage requirements, which amount to approximately 150k CPU cores and 200 PB of storage, ALICE uses the WLCG [2] distributed Grid. The ALICE data are spread over 70 distinct storage elements and stored in about 10 billion files of various sizes, hosted in computing centres worldwide.

The efficient operation of the disk storage is a key element of WLCG. Although many efforts have been made to assure adequate monitoring and status notification, the sheer data volume makes it prohibitively expensive to continuously verify the integrity of each file. As a result, a storage may be functioning as a unit, however certain data may not be accessible or may become corrupted. This usually results in the jobs aborting and complex payloads not completing. In this work, we describe a distributed file crawler that accesses data on a time-cyclic schedule with a quasi-random pattern. It also gathers statistics like the number of files that are corrupted or inaccessible as well as the throughput and download latency of individual storage elements. During the data crawling, each possible exception is caught

and stored for further statistical analysis, together with a description of the technical issues that may have caused the problem and a possible remedy.

A file is considered corrupted in two basic cases: MD5 sum or apparent size difference of the read file with the one stored in the ALICE Grid catalogue. A particular case is that of zero sized files, likely due to metadata inconsistencies, and is thus considered separately in the crawler. Corruption can be due to multiple causes: network error, software bugs, file system or hardware failure and it is particularly hard to defend against. For example, an exception that occurs during a file upload and is not detected may cause partial data writes which can lead to data inconsistencies. When the data are detected and labeled as corrupted it should either be discarded or replaced with a non corrupted replica. Since a single corrupted data file in an analysis workflow can cause the loss of results from many other files processed by the same job, discarding the affected file improves the overall operating efficiency.

A complete check of the disk-based content would take an estimated eleven days to execute. That estimate takes into account the 90 PB of data stored on disk SEs and an aggregated read speed of 100 GB/s, currently observed when analysis jobs saturate the I/O resources. Pausing the data processing for so long is not an option for the experiment and periodically repeating this kind of exercise is out of question. Our approach takes a fraction of this bandwidth (1% of the number of jobs and therefore of I/O bandwidth) to continuously gauge the operational status of the underlying storage nodes. Persistent and transient errors are detected and allow site administrators and Grid operators to take corrective actions. The focus is on response time and having a short delay between when a problem develops (i.e. a storage node is not accessible any more) and when the alert is raised. It is important to the experiment because it ensures a continual high availability of the data sets.

2 Related elements of the ALICE Grid software

One of the requirements of the file-crawler system is that it must be integrated with the existing software solutions that are used within the ALICE project and reuse as many components as possible. JAliEn [3, 4] stands for Java ALICE Environment and it is a software that allows users to execute jobs on the Grid and to read or upload data to storage elements via an API.

The File Catalogue [5] is based on a MySQL cluster, deployed with a master-slave architecture. It stores the metadata of all files stored in the storage elements, like physical location (Physical File Name - PFN), owner, time of creation, size, type, checksum. Each stored file PFN has a unique identification number that is randomly generated, called GUID (Global Unique Identifier). Each file also has a human-readable visible name, logically framed in a LFN (Logical File Name). A basic principle in the ALICE storage protocol is that files stored on the Grid are immutable. Since the content, after the initial file upload, is not allowed to change, the size and checksum are associated with the LFN entries.

XrootD [6] is a data access system developed by the Stanford Linear Accelerator Center and used extensively by ALICE. It provides an API similar to Posix for accessing any type of files and directories. XrootD features a communication protocol, based on TCP, between the client and the XrootD server.

LPM (Lightweight Production Manager) is an ALICE workflow engine for Grid job management and is integrated in the central Grid monitoring and accounting service *alimonitor.cern.ch*. LPM can perform automatic job submission, based on Grid load, user and production priorities. It supervises the job execution and allows for dependencies

between job types, which is used to create job chains where automatic submission of a next chain entry depends on the completion of a previous job.

3 Architecture of the file crawler system

In order to detect corrupted files and analyse the health and performance of storage elements (SEs), we have developed a file crawler, which periodically submits Grid jobs targeted at the computing element(s) closest to the analyzed SE. Jobs are launched in parallel for all SEs and receive as input a list of PFNs extracted from the File Catalogue known to be present on the target storage element and produce two output files: one with general statistics about the crawling and another one with information about each file analysed. After the results are obtained, the monitoring database is updated with the latest computed values. The jobs are launched programmatically via predefined and parameterized JDL (Job Description Language) files built at runtime. If function arguments must be passed, they can be added in JDL in the *Arguments* field. Figure 1 presents the general architecture of the file crawler. The six main execution phases (startup, cleanup, crawling prepare, crawling, merging and database update) are detailed below.

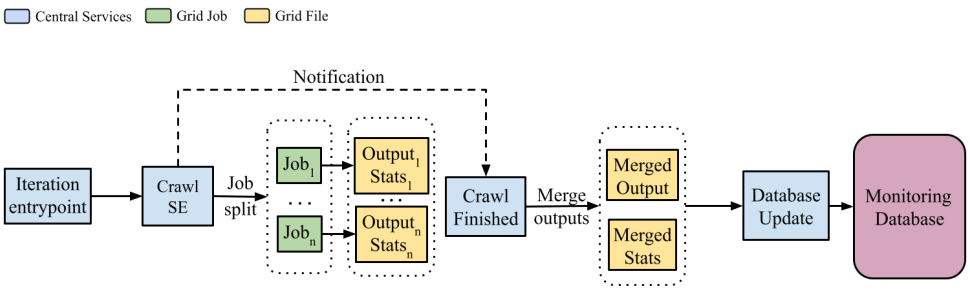


Figure 1. The architecture of the file crawler.

3.1. Crawler timestamps and execution steps

The crawler is started by an LPM job that fully manages the lifecycle. A timestamp is associated with the moment in time in which the crawling process starts. This timestamp will be later referred to in the paper as an 'iteration timestamp'. Every time the crawler starts a new execution cycle it is considered to start a new 'iteration'. The decision of starting a new iteration is taken by examining the difference between the current timestamp and the timestamp of the previous iteration as well as whether it was successfully executed or not. Every iteration of the crawler is associated with a subset of storage elements to be analysed and a new data set to be examined.

3.2 Cleanup

A crawler job goes through multiple states: 'inserting', 'waiting', 'assigned', 'started', 'running', 'saving' and 'done'. A job is considered complete when it enters the 'done' state. There may be situations when the job cannot complete before the next iteration starts or when it is blocked in a waiting state, preventing the rest of the crawling process from

continuing. Factors like network congestion, server load, hardware or software failures are predominant in such situations. These affect about 2% of the tests. The cleanup step that is performed by the crawler consists of forcefully terminating jobs from previous iterations that could not finish. JDL fields like 'MaxWaitingTime', can be used to automatically kill a job if it stays in the waiting state for too long. Another type of cleanup that is executed by the crawler is the deletion of files written by the crawler to the Grid that are older than one month.

3.3 Crawling prepare

In order for the crawling process to start, the number of jobs that will be launched and the sample size of PFNs that will be extracted during the crawling process must be determined. Both of these values are computed based on the formula which is explained in detail in section 4.1.

To ensure that files are accessed in a quasi-random manner, a new function called *getRandomPFNs* was added to both the API and the command line interface provided by JAliEn. It has two parameters: the storage element number from which to get the files and the number of PFNs to extract. The function executes a SQL query which sorts the results using the 'rand' function. The Dispatcher object is used to execute the function remotely on the central services, since it requires access to CERN's central services. Because the query can be quite resource-intensive and time consuming, the result of the function is stored in an in-memory array. This array is then cached to a file on disk and fed to the crawling master job using the *InputFile* JDL tag. Random file extraction and list serialization are database-intensive, therefore, the aforementioned computations are done independently for every storage element in separate parallel jobs. If the list of SE was iterated on sequentially, the computations would have delayed the launch of crawling jobs for the storages at the end of the array.

3.4 Crawling process

The crawling process starts from a master job that is launched programmatically and it is then splitted into multiple subjobs. Every masterjob reads the serialized input file, decodes it and extracts a chunk of that file. The chunk is delimited by 2 indexes in the array which are computed in the crawling prepare phase and sent as arguments. The resulting chunk of data represents the PFNs that must be analysed by the subjob for a specific storage element.

Firstly, for each PFN a specific set of xrootd commands (*xrdfs stat)* [8] to get the file attributes directly from the SE is used to identify problems with the underlying data server, for example if the files are inaccessible. All possible exceptions that can cause the call to fail are identified and split into multiple categories (examples are shown in section 4.3). An error during this step causes the rest of the tests for that file to be skipped.

Secondly, the storage element is analyzed for corrupted files using the MD5sum stored in the Grid catalogue at the time the file was written. Corruption of that file is detected by recomputing its checksum and comparing it with the original. There are two crawling strategies that can be used: the files are downloaded from the remote storage to the current working directory of the subjob and the second where the checksum is computed remotely by the storage. The second choice is better as it avoids file download over network, reducing I/O. Its use is limited because XrootD does not support file checksum recomputation on the remote side yet. This feature will be implemented and added to XrootD in the future, but is already fully supported by the crawler system.

Every subjob writes the results in two separate files stored on disk, 'output' and 'stats'. The first file contains the result of the analysis for each PFN, including specially defined error codes and the second file represents the general statistics that are gathered during the crawling process. The file which stores the statistics is in a JSON format, while the output file can either have a JSON or CSV format, depending on a user-provided parameter to the crawler.

3.5 Merging

Every crawling subjob produces its own output files. To have a better view of the data, all output files from each storage element are merged into a single output file. The merging process is started by LPM when crawling subjobs finish. The resulting file per storage element is uploaded to the crawler's home directory and its name is prefixed with 'merged'. The merged output files are subsequently processed to analyze the health of a particular storage.

3.6 Database update

The merged data are stored as Grid files in the home directory of the crawler. Even though these files can be read and interpreted directly by system administrators, it is more useful to upload the results to a central monitoring database with a common visualization interface. To that end, the merged files are imported in several database tables. Importing data to the databases is done following a method triggered automatically by LPM when the merging finishes. The process runs on the central services at CERN to avoid remote connections to the database.

4 Implementation details

The crawler code is implemented in Java and it is integrated in two separate projects: JAliEn and alimonitor-lib. A high-level overview of the architecture is presented in the previous section. In this section we describe a few implementation details. Since xrootd-enabled storage elements are now predominant in WLCG, the methods and implementation of the crawler are relatively easy to re-apply for the majority of the organizations using the WLCG infrastructure. We estimate that the largest efforts to adopt this tool will be the coupling with one particular element - the Grid file catalogue, which tends to be a specific implementation across the organizations.

4.1 Sample size calculation

As already stated above, periodic analysis of all files on the ALICE storage elements is prohibitively expensive. In such situations, using representative sampling is the preferred method.

Moore and McCabe state in the book "Introduction to the Practice of Statistics" [7] that "random sampling eliminates bias by giving all individuals an equal chance to be chosen". Random sampling can be easily achieved in software and in our case, we use the "rand" function provided by SQL to randomly select files from the File Catalogue.

To compute the number x of files to be extracted per storage element (see formula 4) and the number of crawling jobs to be spawned in each iteration, we have chosen a function

that scales linearly with the number of files of a storage element. Formula (1) shows a linear function that is defined on a closed interval [a, b], representing a range that spans from 0 to the number of files of a storage element and whose values are bound by a predefined closed interval [c, d], specified during crawler configuration.

$$f : [a, b] \rightarrow [c, d],\ f(x) = mx + n \tag{1}$$

$$f(a) = ma + n = c \tag{2}$$

$$f(b) = mb + n = d \tag{3}$$

By subtracting equation (2) from equation (3) we can also find the slope of the function, m, in terms of the known variables a, b, c, d. By substituting the result in either (2) or (3) we can find the value of the intercept, n. The final formula for the linear function is given in (4).

$$f(x) = \frac{d - c}{b - a}(x - a) + c \tag{4}$$

The function is linear and monotonically increasing. It is defined by the minimum and the maximum number of files from any storage and has values in the default interval (100, 10 000), since we want to bound the number of files extracted to a predefined range in order to have a low impact on the I/O and site resources.

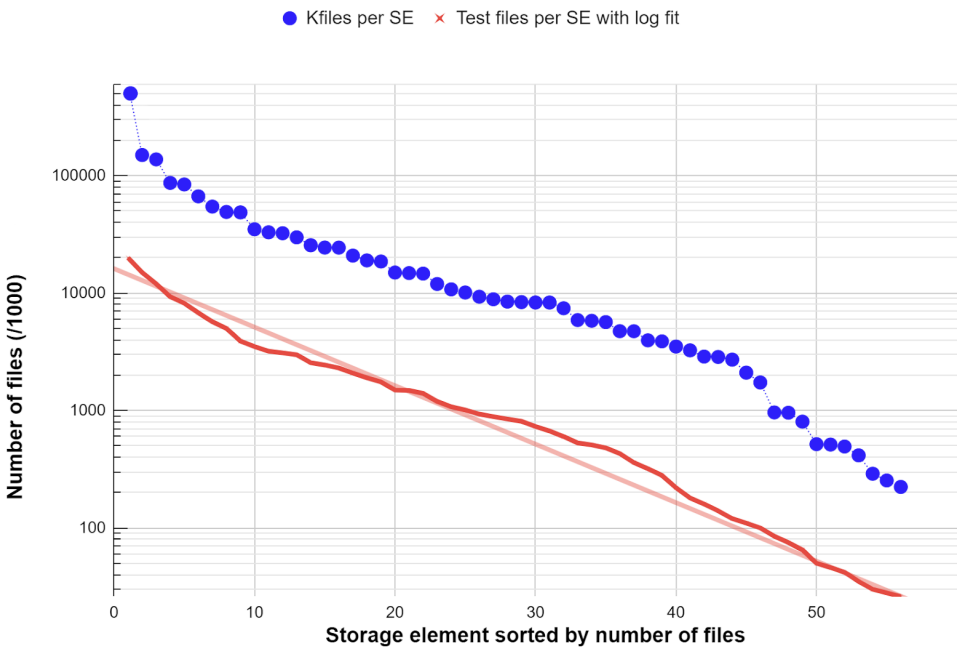


Figure 2. Total number of files stored in the SEs and the corresponding SEs test sample size in number of files. The largest SE contains 600 million files, the smallest 200 thousand files.

The number of files in a SE depends on the storage capacity. The distribution of files in the storages ranges from 600 million files in CERN-EOS to 200 thousand files in PNPI-SE. Figure 2 shows the distribution of files among all SEs used in ALICE along with the number of test files to analyse per SE in each iteration. There is a proportionality between the number of test files and the total file count in the SE which allows us to have equally

accurate test results for all SEs. Upgrades in the capacity of a SE will be picked up by the crawler and the sample size will be automatically updated, as it computes the test size for all SEs before each iteration.

4.2 Data gathered by the crawler

Each iteration 'merged_output' file contains for each file:

- Unix timestamp for the start of the iteration;
- ID of the storage element on which the PFN resides;
- PFN and the corresponding GUID;
- file size and MD5sum after file download and the corresponding Grid catalogue values;
- milliseconds to execute *xrdfs* and download the file;
- an enum string pointing to the supertype of the result of the crawling process: 'ok', 'corrupted', 'inaccessible', 'internal error' or 'unknown error';
- the error message that caused the PFN analysis to fail (set to *null* if there is no error);
- a string pointing to the status code of the crawling process and the timestamp at which the file was generated.

The 'merged_stats' file contains information about the crawling process for a specific iteration and storage element. The following fields are stored:

- Unix timestamp for the start of the iteration;
- ID of the storage element on which the analyzed PFN resides;
- average download throughput in MB/s;
- total duration of the crawling process in milliseconds;
- total size in bytes for all PFNs downloaded and the timestamp of the file generation.

4.3 Status codes overview

The crawler collects fault causes and groups them by category and SE. This allows us to understand what type of errors are encountered during the crawling process and which ones are the most common. A list of example status codes and their description is presented below:

- *S_FILE_CHECKSUM_MATCH*, when the checksum of the downloaded file matches the checksum registered in the catalogue. This is returned if all internal checks passed;
- *MD5_CHECKSUMS_DIFFER,* when the file could be read from the storage and the size matches, but the content has a different MD5 checksum than the expected one;
- *E_CATALOGUE_MD5_IS_BLANK*, when the MD5 registered in the catalogue is blank;
- *E_GUID_NOT_FOUND*, when the GUID cannot be retrieved from the PFN;
- *E_PFN_NOT_READABLE*, when the PFN access token cannot be filled;
- *XROOTD_EXITED_WITH_CODE*, when *xrdcp* exited with non-zero code, but no other details could be inferred from the command output;
- *E_PFN_DOWNLOAD_FAILED*, when the file download process failed;

- *NO_SERVERS_HAVE_THE_FILE*, when the server couldn't locate (by broadcasting) any replica of the requested file;
- *E_FILE_EMPTY*, when the size of the downloaded file is 0 bytes;
- *XROOTD_TIMED_OUT*, when the transfer took longer than allowed (with a value of 20 kB/s + 60 s overhead) so *xrdcp* was forcefully terminated;
- *NO_SUCH_FILE_OR_DIRECTORY*, when the server returned an authoritative answer that the file is not present anywhere in its namespace;
- *LOCAL_FILE_CANNOT_BE_CREATED*, when for permissions or space reasons, the target local file could not be created.

5 Crawler data analysis

Data gathered during the crawling process is used to create visualisations in MonALISA, which help system administrators analyse SE's performance and health, and start the debugging process if issues are encountered.

5.1 Status codes analysis

Status codes extracted from the crawler are inserted into the monitoring database from a data producer deployed in the central services, which pushes data into time series format. This analysis allows us to detect what type of faults occur during the crawling process and inspect the overall state of SEs over time.

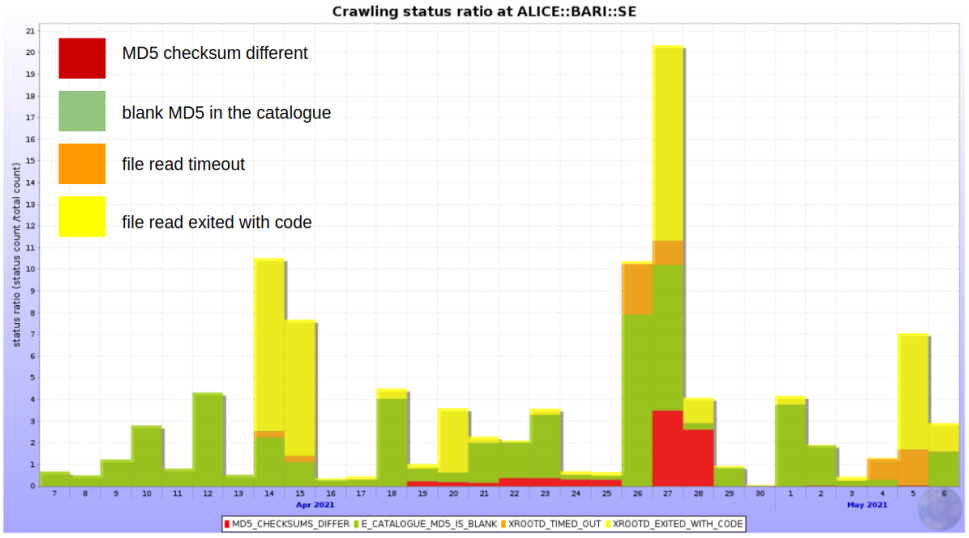


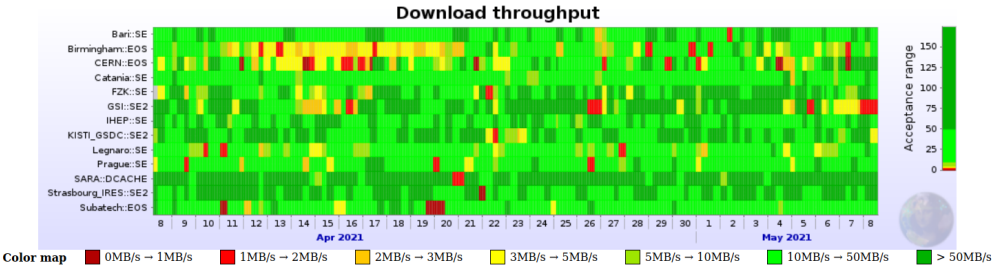
Figure 3. Status codes detected by the crawler in a time frame of one month for Bari SE. The Y axis represents the status ratio, which is defined as the status code count divided by the total count of all status codes.

Figure 3 shows the status codes detected by the crawler for the Bari SE in a one month time frame. For certain periods of time, up to 3% of the sample size is detected as corrupted. However, in general, there is a low percentage of really corrupted data. Blank MD5 in the catalogue status codes are automatically fixed by the crawler, as it updates the

missing value with the one recomputed after file download. Read timeouts and read exiting with non-zero codes usually occur during file copy and are heavily influenced by the SE load.

5.2 Throughput analysis

Throughput is computed by the crawler during each file download. It helps with the performance analysis of SEs. For a specific time period, throughput is computed as an average of all values registered in that interval, weighted with the number of files analysed. Figure 4 shows the distribution of download throughput for multiple SEs for a one month period. All throughput values below 5 MB/s are considered below optimal, and in the image they are represented with shades of yellow and shades of red. The typical cause for low throughput values is heavy analysis load. In the figure, there are a few days where Birmingham:EOS had below optimal throughput, but overall the average throughput is optimal for all SEs analysed in that time frame. Among all storages analysed in this snapshot, SARA:DCACHE, which is a Tier 1 SE, has the highest download throughput average of above 95 MB/s.



Statistics

Link name	Data		Average
	Starts	Ends	Download throughput
Bari::SE	08 Apr 2021 05:14	08 May 2021 09:23	39.97 MB/s
Birmingham::EOS	08 Apr 2021 05:15	08 May 2021 09:21	14.68 MB/s
CERN::EOS	08 Apr 2021 05:19	08 May 2021 09:43	27.37 MB/s
Catania::SE	07 Apr 2021 23:03	08 May 2021 09:25	26.80 MB/s
FZK::SE	08 Apr 2021 11:25	08 May 2021 09:21	45.23 MB/s
GSI::SE2	08 Apr 2021 05:13	08 May 2021 09:14	58.79 MB/s
IHEP::SE	08 Apr 2021 05:15	08 May 2021 09:23	44.28 MB/s
KISTI_GSDC::SE2	08 Apr 2021 05:17	08 May 2021 09:08	46.00 MB/s
Legnaro::SE	08 Apr 2021 05:16	08 May 2021 09:06	23.10 MB/s
Prague::SE	08 Apr 2021 05:17	08 May 2021 09:18	26.75 MB/s
SARA::DCACHE	08 Apr 2021 05:18	08 May 2021 09:19	95.15 MB/s
Strasbourg_IRES::SE2	08 Apr 2021 05:16	08 May 2021 09:10	49.27 MB/s
Subatech::EOS	08 Apr 2021 05:15	08 May 2021 09:18	32.43 MB/s

Figure 4. Download throughput registered during a one month period for several storage elements, along with the average throughput values for the same time interval.

5.3 PFN sample analysis

The analysis results for all PFN samples are written into the monitoring database and visualised in MonALISA. While status codes analysis gives an overview of the errors and their type for a specific SE, samples analysis allows system administrators to trace the error back to the PFN, analyse the full text error message and reproduce the error by re-executing the command that failed. Figure 5 shows examples of PFNs analysed by the crawler and their output. PFNs can be filtered by status code, SE name and size. The full text message is displayed in a tooltip and contains the text message of the error and the full command that

caused the crawling process to fail. For example, the displayed PFNs from Clermont SE resulted in a state where *xrdfs* could not confirm the upload of files. In the future, we want to add in the webpage debugging tips for system administrators per PFN and possible solutions to common problems, as it would save debugging time.

		MD5_CHECKSUMS_DIFFER,XRDFS_C			Filter
PFN	SE Name	Status Name	Size (B)	Timestamp	Status Mes
root://neos.nipne.ro:1094//08/26705/b54354f8-1227-11e8-8c69-2ba5bf90fcb5	ALICE::NIHAM::EOS	MD5_CHECKSUMS_DIFFER	98401595	Apr 22,2021 06:49	Local file doesn\'t match ca
rinr03.inr.troitsk.ru:1094//13/61373/98f6d712-42ee-11e0-8740-0019bbc6247c	ALICE::TROITSK::SE	MD5_CHECKSUMS_DIFFER	197750808	Apr 23,2021 09:35	Local file doesn\'t match ca
//ali-nxrd.to.infn.it:1094//03/59050/73e90802-757d-11e0-8a17-b7d2e3473590	ALICE::TORINO::SE2	MD5_CHECKSUMS_DIFFER	53057140	Apr 14,2021 19:08	Local file doesn\'t match ca
s-mgm01.sdfarm.kr:1094//03/00517/358bab02-848d-11e7-b871-cfae48f7be88	ALICE::KISTI_GSDC::EOS	MD5_CHECKSUMS_DIFFER	130366274	Apr 19,2021 17:59	Local file doesn\'t match ca
root://neos.nipne.ro:1094//08/21339/846c57cc-c035-11e9-b56a-27fe2bbbf9cc	ALICE::NIHAM::EOS	MD5_CHECKSUMS_DIFFER	12211	Apr 14,2021 19:04	Local file doesn\'t match ca
/clralicexrd.in2p3.fr:1094//04/12325/a19824f6-f1c6-11e6-ba09-ebd157cb4b4a	ALICE::CLERMONT::SE	XRDFS_CANNOT_CONFIRM_UPLOAD	279128250	Apr 14,2021 19:06	[/cvmfs/alice.cern.ch/x86_64-2
/clralicexrd.in2p3.fr:1094//01/29766/831e1676-f1c5-11e6-b0df-138910a3cca0	ALICE::CLERMONT::SE	XRDFS_CANNOT_CONFIRM_UPLOAD	87568203	Apr 14,2021 19:07	[/cvmfs/alice.cern.ch/x86_64-2
/clralicexrd.in2p3.fr:1094//04/55516/b9b4994c-efd3-11e6-8e73-d76fe48573a0	ALICE::CLERMONT::SE	XRDFS_CANNOT_CONFIRM_UPLOAD	3274	Apr 14,2021 19:08	[/cvmfs/alice.cern.ch/x86_64-2
/clralicexrd.in2p3.fr:1094//04/35439/2d603a12-f1c4-11e6-8b5a-b3cae7df7731	ALICE::CLERMONT::SE	XRDFS_CANNOT_CONFIRM_UPLOAD	35667565	Apr 14,2021 19:09	[/cvmfs/alice.cern.ch/x86_64-2
clralicexrd.in2p3.fr:1094//01/16208/89beaa56-ed4f-11e6-b979-9b28e76b708d	ALICE::CLERMONT::SE	XRDFS_CANNOT_CONFIRM_UPLOAD	554273	Apr 14,2021 19:07	[/cvmfs/alice.cern.ch/x86_64-2

Figure 5. A list of PFN samples already analysed by the crawler.

6 Conclusion

In this work, we present a distributed file crawler system that analyses the condition of the storage elements used by ALICE. It downloads random statistically significant samples of files from every storage, recomputes checksums in order to detect file corruption, detects the cause and type of the errors, gathers statistics during the process and writes the data into files on the Grid and into the database for persistence and visualisation in MonALISA. The crawler is written in Java and it is fully integrated with the existing ALICE software stack. As of May 2021, there are more than 11 million PFNs already analysed by the crawler. Next, we plan to add more fine-grained performance analysis, early fault detection of storage issues, suggest actions and debugging tips for system administrators and heavy users and use the collected data to optimize the analysis patterns, such as spreading the analysis tasks in time to avoid overloads or combine analysis tasks with higher and lower I/O demands.

7 References

1. *ALICE collaboration website*, https://alice-collaboration.web.cern.ch/, accessed: 2021-06-15

2. I. Bird, Computing for the Large Hadron Collider, Annual Review of Nuclear and Particle Science, 61: 99–118 doi:10.1146/annurev-nucl-102010-130059 (2011)

3. C. Grigoras, A. G. Grigoras, et al., JAliEn–A new interface between the AliEn jobs and the central services. Journal of Physics: Conference Series. vol. **523**. No. 1. IOP Publishing (2014)

4. M. Martinez, C. Grigoras, V. Yurchenko, JAliEn: The New ALICE High-Performance and High-Scalability Grid Framework. EPJ Web of Conferences, vol. **214** (2019)

5. S. Schreiner, S. Steffen, et al., Securing the AliEn File Catalogue - Enforcing Authorization with Accountable File Operations. Journal of Physics: Conference Series, vol. **331**, no. 6 (2011)

6. A. Dorigo, P. Elmer, F. Furano, A. Hanushevsky, XROOTD-A Highly scalable architecture for data access, *WSEAS Transactions on Computers* 1.4.3 (2005)

7. A. Craig, S. Moore and P. McCabe, *Introduction to the Practice of Statistics*, Third edition, p. 219 (2006)

8. *xrdfs command reference manual*, https://xrootd.slac.stanford.edu/doc/man/xrdfs.1.html, accessed: 2021-06-15