EPN2EOS Data Transfer System

Alice Florența Șuiu[1,2,*], *Costin* Grigoraș[1,**], *Sergiu* Weisz[1,2,***], and *Latchezar* Betev[1,****]

[1]CERN, Esplanade des Particules 1, 1211 Geneva 23, Switzerland

[2]Politehnica University of Bucharest, Splaiul Independenței no. 313, sector 6, Bucharest, Romania

Abstract. ALICE is one of the four large experiments at the CERN LHC designed to study the structure and origins of matter in collisions of heavy ions and protons at ultra-relativistic energies. To collect, store, and process the experimental data, ALICE uses hundreds of thousands of CPU cores and more than 400 PB of different types of storage resources.

During the LHC Run 3, started in 2022, ALICE is running with an upgraded detector and an entirely new data acquisition system (DAQ), capable of collecting 100 times more events than the previous setup. One of the key elements of the new DAQ is the Event Processing Nodes (EPN) farm, which currently comprises 250 servers, each equipped with 8 MI50 AMD GPU accelerators. The role of the EPN cluster is to compress the detector data in real-time. During heavy-ion data taking the experiment collects about 900 GB/s from the sensors, compressed down to 100 GB/s, and then written to a 130 PB persistent disk buffer for further processing. The EPNs handle data streams, called Time Frames, of 10 ms duration from the detector independently from each other and write the output, called Compressed Time Frames (CTF), to their local disk. The CTFs must be transferred to the disk buffer and removed from the EPNs as soon as possible, to be able to continue collecting data from the experiment. The data transfer functions are performed by the new EPN2EOS system that was introduced in the ALICE experiment in Run 3. EPN2EOS is highly optimized to perform the copy functions in parallel with the EPN data compression algorithms and has extensive monitoring and alerting capabilities to support the ALICE experiment operators. The service has been in production since November 2021. This paper presents the architecture, implementation, and analysis of its first years of utilization.

1 Introduction

EPN2EOS serves as a vital tool responsible for managing the entire raw data flow of the ALICE experiment [1]. Positioned between the detector readout and the persistent remote storage (EOS), EPN2EOS operates its tasks within a set of constraints imposed by the experiment.

*e-mail: asuiu@cern.ch

**e-mail: costin.grigoras@cern.ch

***e-mail: sergiu.weisz@upb.ro

****e-mail: latchezar.betev@cern.ch

Foremost, EPN2EOS must guarantee the preservation of all information contained within the files generated by the ALICE experiment. This preservation is critical to avoid any adverse impact on the results obtained from the experiment.

The files are initially stored on the Event Processing Node (EPN) local disk with a 4 TB capacity. This storage space is sufficient for approximately 3 hours of data taking. Thus, EPN2EOS faces the imperative of achieving fast data transfer in order for the ALICE experiment to continue collecting data.

Since EPN2EOS operates on the EPNs, it must respect constraints on available computing resources. By optimizing its resource usage, EPN2EOS enables the EPNs to dedicate the majority of their resources to efficiently collecting, compressing, and storing the data received from the experiment.

Lastly, EPN2EOS bears the responsibility of ensuring data integrity during the transfer process. To accomplish this, it calculates a checksum for each file, thus verifying the data integrity throughout the entire transfer operation.

2 EPN2EOS in the data transfer path

This section presents an overview of the system architecture that is schematically shown in Fig.1.

The ALICE experiment generates a substantial amount of data, which is collected by the First Level Processor (FLP) nodes. This FLP cluster comprises 200 nodes, responsible for reading, aggregating, and transmitting data from the detector to the EPNs [2]. The FLP farm receives data from the detectors at a total rate of 3.5 TB/s over 8000 optical links. They perform a first level of data compression to 900 GB/s by zero suppression. In addition, they have the possibility to perform calibration tasks based on local information from the part of the detector they serve. After all FLPs have built their Sub-Time Frames (STF) of an individual Time Frame (TF), an available EPN is selected and all STFs are sent there and the full TF is built.

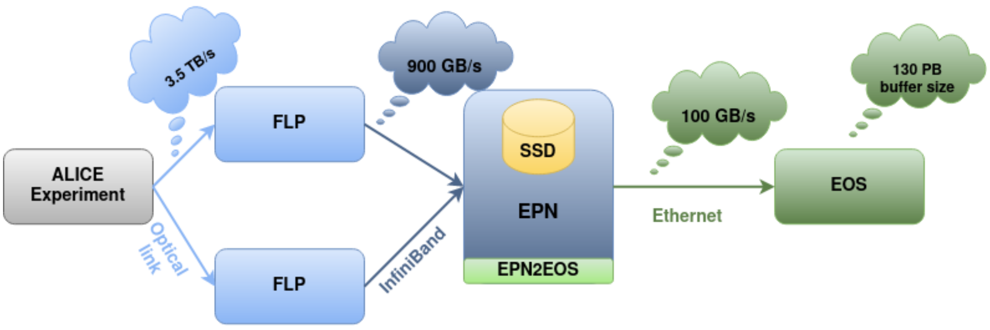


Figure 1: EPN2EOS in the data transfer path

The EPN farm consists of 250 computing nodes, each with 2 AMD Rome 32-core CPUs and 8 AMD MI50 GPUs with 32 GB memory. It relies on an InfiniBand physical network to transport time frame data, enabling real-time communication between FLPs and EPNs [2]. Once an EPN has received a complete time frame, it has 30 seconds to process it. This involves interpreting the raw data, applying calibration, running reconstruction, and assembling compressed time frames. The compressed data is then stored on its local SSD before being copied to storage.

The storage solution used, known as EOS [3], boasts low latency and high capacity. It effectively accommodates the storage of vast amounts of data in files and facilitates interactive analysis. The EOS comprises two main parts: the client-side, which offers a command-line interface and access to a mounted file system, and the server-side, consisting of components for storing file metadata and actual data, as well as a message queue for asynchronous message transmission [4].

The vital task of managing experiment data and their transfer to the storage (EOS) is handled by EPN2EOS, which operates on each EPN.

If any component in Fig. 1 fails, then data collection is stopped and the problem is investigated. To prevent such situations, the entire system has alert capabilities to signal a possible malfunction in time. Also, during the entire period of data collection, 24/7 shifts are carried out with expert personnel who can take immediate measures in case of errors.

3 Implementation details

This section presents the implementation details for accomplishing the system objectives mentioned in Sec. 1. EPN2EOS maintains an internal queue of files to be transferred, reacting to filesystem events triggered by new metadata files created on the local disk. A metadata file is associated to a specific data file and is located in a specified directory for each EPN node and contains the attributes given in table 1. There must be a unique metadata file for each data file. The creation of these files triggers the subsequent copy of the data file from the EPN to the storage. The metadata file contains mandatory and optional parameters marked with red and green in Tab. 1. The filename must be unique and have the suffix ".done".

Table 1: Metadata file content

Metadata Attribute	Description	Type
LHCPeriod	LHC data taking period + detector name, i.e. `LHC22o` or `LHC23a_TPC`	required
run	a number that marks a continuous time frame in which data is taken, i.e. `543512`	required
lurl	the local EPN path to the data file	required
type	`raw`, `calib`, or `other`; default is `other` (1)	optional
ctime	default the timestamp of the *lurl* file, in Unix epoch seconds	optional
size	default the size of the *lurl* file, in bytes	optional
guid	default is an auto-generated version 1 UUID	optional
surl	the remote storage path where the data file is stored (2)	optional
md5	default the checksum of the *lurl* file; only filled after a successful transfer, if needed (2)	optional
xxhash	default calculated from the *lurl* file, only filled after a successful transfer, if needed (1), (3)	optional
seName	default is taken from the configuration file, currently `ALICE::CERN::EOSALICEO2` (4)	optional
seioDaemons	default is taken from the configuration file, currently `root://eosaliceo2.cern.ch/` (4)	optional
priority	`low` or `high`; default is `low` (5)	optional
persistent	number of days that the data is available on storage; default is *forever* (6)	optional
det_composition	contains a list of detectors, i.e. `"ITS,TPC,TRD,V00"` (7)	optional

(1) Type can only be raw, calib, or other. The raw data contains information about detector signals from each collision for physics reconstruction, while calib contains data for detector calibration.

(2) If md5, xxhash, and surl are not present in the metadata file, EPN2EOS computes them.
(3) The xxhash algorithm is used to check the integrity of the data after transfer.
(4) The seName and seioDaemons represent attributes through which EPN2EOS can connect to the remote storage.
(5) Each data file has an associated priority (low or high) specified in the metadata file. If the priority is missing from the metadata file, it will be considered low as default.
(6) This metadata key corresponds to the availability of the data file on the storage in units of days. After this period the data file will automatically be deleted. If the persistent attribute is missing from the metadata file, then the data file will not expire from storage.
(7) det_composition: If this metadata contains a single detector, then EPN2EOS appends the detector name to the period name. If there is more than one detector or this metadata key is missing, then the run is considered global, and only the period name is used.

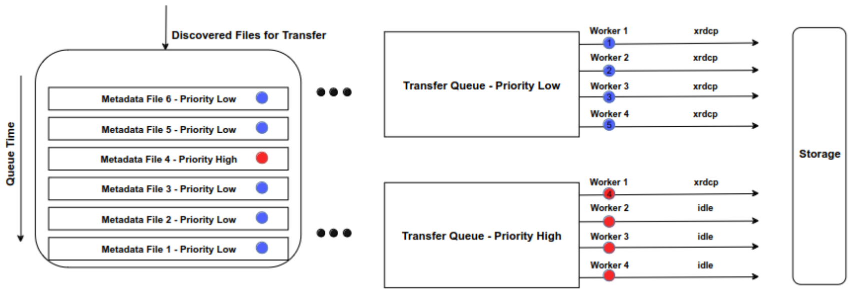

Discovered Files for Transfer
Queue Time
Metadata File 6 - Priority Low
Metadata File 5 - Priority Low
Metadata File 4 - Priority High
Metadata File 3 - Priority Low
Metadata File 2 - Priority Low
Metadata File 1 - Priority Low
Transfer Queue - Priority Low
Transfer Queue - Priority High
Worker 1
Worker 2
Worker 3
Worker 4
xrdcp
idle
Storage

Figure 2: Watcher and data transfer diagram

All the metadata files are placed in a directory configured when EPN2EOS starts. A file event watcher is created on this directory, a process that tracks activity on the metadata location directory. Its primary function is to notify EPN2EOS when a new metadata file has been added to the directory. Each data file has an associated priority, and for each priority, EPN2EOS creates a separate queue in which it puts the corresponding files. In addition, the tool creates separate threads to handle the transfer of files from the priority queue. The default size of each thread pool is 4, but it can be configured via the configuration file. The data path of the watcher and EPN2EOS is illustrated in Fig. 2. In this example, only one file with high priority is discovered. Therefore, the associated data file is transferred by one of the workers assigned to the high-priority queue. As there are no other files with high priority, the other three workers remain in an idle state. At the same time, there are five files with low priority and only four workers available for transfer, so one of the files remains in a waiting state. The implementation logic for the watcher is illustrated in algorithm 2 and for the tool in algorithm 1.

EPN2EOS uses the XRootD [5] protocol, which is a data transfer protocol integrated and optimized to transmit files very quickly and efficiently. It is a robust protocol that offers scalability in communicating with several clients.

EPN2EOS uses the xxhash [6] algorithm to assure the integrity of the data transfer between the EPN and EOS. This checksum algorithm is fast, allows parallel processing of data blocks, and is implemented in EOS [7]. The xxhash value is either put in the metadata file by the upstream processing or is calculated by EPN2EOS. The return value from the transfer command is compared to the metadata file value. So, the EOS attributes on the target directory structure require the use of the xxhash checksum algorithm. If the two obtained xxhash values are equal, then the md5 value is computed, if not already in the metadata file. Although EPN2EOS uses the xxhash algorithm for faster data integrity check after the transfer, for all subsequent file accesses, the md5 algorithm is used to guarantee integrity. All other services are using as default the md5 algorithm for each file access. Thus, to maintain compatibility with other services, we had to keep the md5 algorithm. However, it could not be used for data transfer due to being too slow and not allowing for parallel processing of data blocks [7].

Once the file transfer and all additional operations are successfully completed, the data file is deleted from the local disk.

In case of a failed data transfer, the copy operation is retried after a delay that is calculated using the Exponential Backoff [8] strategy. Thus, the delay time increases exponentially with the number of attempts to transmit the file (2^1 (second attempt), 2^2 … *maxBackoff* (60 seconds)) and after this delay the metadata file is added back to the transfer queue. The maximum limit of the Exponential Backoff strategy is also specified via a configuration file. The implementation logic for the delay computation can be seen in algorithm 3.

Algorithm 1 EPN2EOS - Transfer File

```
1: Given : md5Enable
2: Input : fileElement
3: Require : $md5Enable$ is Boolean
4: $status \leftarrow xrootd(fileElement.srcDataPath, fileElement.destDataPath)$
5: if $status == success$ then
6:     if $(md5Enable == true) \wedge (fileElement.md5 == null)$ then
7:         $computeMD5(fileElement)$
8:     end if
9:     $delete(fileElement.srcDataPath)$
10: else
11:     $computeDelay(fileElement)$
12:     $addToTransferQueue(fileElement)$
13: end if
```

Algorithm 2 Watcher - Process File

```
1: Given : metadataDir, event
2: Require : $metadataDir$ is String, $event$ is WatchEvent
3: if $receiveNotification(metadataDir, event)$ then
4:     if $event.file.name.endsWith == ".done"$ then
5:         $fileElement \leftarrow readMetadata(event.file)$
6:         $addToTransferQueue(fileElement)$
7:     end if
8: end if
```

Algorithm 3 EPN2EOS - Compute Delay for File

```
1: Given : maxBackoff
2: Input : fileElement
3: Require : maxBackoff is Unsigned Integer
4: $fileElement.nrTries \leftarrow fileElement.nrTries + 1$
5: $delayTime \leftarrow min(2^{fileElement.nrTries}, maxBackoff)$
6: $fileElement.time \leftarrow currentTimeMillis() + delayTime * 1000$
```

4 Results

This section presents the results obtained after EPN2EOS was put into production. EPN2EOS runs as a daemon service on EPNs since November 2021, after the upgrade of the ALICE experiment. Since then and up to the writing of this article (2023-08-01), EPN2EOS has transferred approximately 150 PB of data (Fig. 3) in about 100 million files, for an average file size of 1.5 GB.

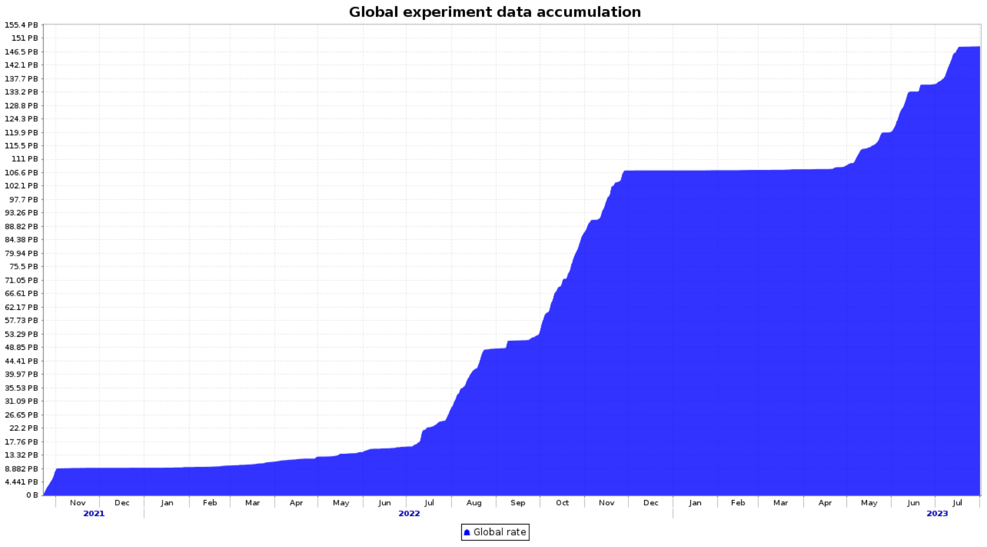


Figure 3: Data accumulation starting with the end of 2021

In the first days of operation, EPN2EOS used 20 transfer threads. With 20 parallel copy operations started by each EPN, a maximum aggregated transfer speed of 27 GB/s was obtained from the EPN farm to the EOS buffer. With fewer concurrent streams and IO operations, a higher transfer rate was achieved. The optimal point was found at 4 transfer threads per transfer service for which a maximum aggregated transfer speed of 42 GB/s was obtained. This is still the default setting for the transfer thread pool size.

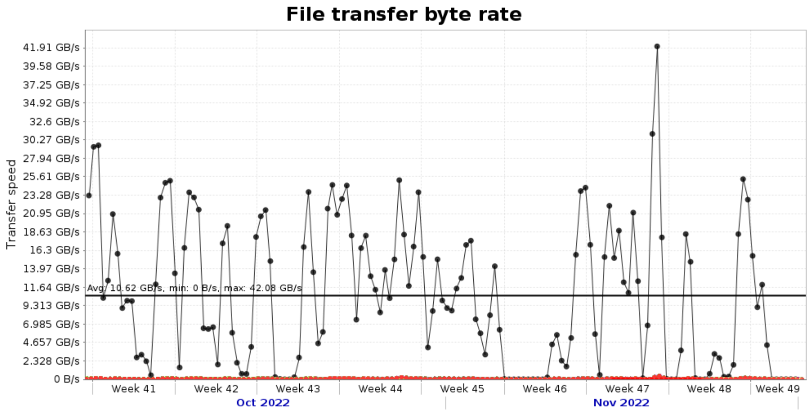
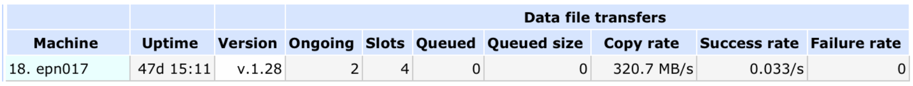


Figure 4: Transfer speed

In Fig. 4 it can be seen how the transfer speed was observed over two typical months of proton-proton data taking in 2022.

During the service operation, there were a few unforeseen cases that were debugged and improved. A particular case was that of intermittent network or storage errors that led to interrupted transfers. These left incomplete files on the target endpoint that cannot be overwritten due to the write-once storage policy, preventing accidental overwrites of experimental data. The solution was for EPN2EOS to keep track of the number of attempts on each file and append this counter to the file name, creating unique paths for each attempted copy operation.

EPN2EOS is a critical component of the ALICE experiment, therefore real-time monitoring as well as detailed logs of its actions are necessary. Internal EPN2EOS metrics as well as system parameters are sent to the MonALISA [9] monitoring framework from where the data can be aggregated for accounting purposes, and displayed to operators. Also, e-mail alerts are sent based on monitored data in case of errors.



			Data file transfers						
Machine	Uptime	Version	Ongoing	Slots	Queued	Queued size	Copy rate	Success rate	Failure rate
18. epn017	47d 15:11	v.1.28	2	4	0	0	320.7 MB/s	0.033/s	0

Figure 5: Entry in the Monitoring page

An example view of the current status of one EPN node is shown in Fig. 5. The meaning of the information presented in this figure can be found in table 2.

Table 2: Monitoring metrics

Metric	Description
Machine	EPN node for which the messages are logged
Uptime	Time that has passed since the last restart of EPN2EOS
Version	Version of EPN2EOS that runs on the EPN node
Ongoing	Number of currently active transfers
Slots	Maximum number of transfers that could run in parallel (per priority queue)
Queued	Number of files in the transfer queue
Queued size	Total size of the files waiting to be transferred
Copy rate	Data transmission rate
Success rate	Success rate, in files/second
Failure rate	Error rate, in files/second

5 Conclusion

In conclusion, EPN2EOS is a fully functional standalone system for data transfer between the ALICE online processing cluster EPN and the IT-managed EOS storage. It is written in Java [10], works in the challenging condition of real-time data taking, and has transfer priority scheduling, robust error handling system, monitoring, and messaging. EPN2EOS is used in production by the ALICE collaboration and represents the only method of data transfer from the experiment to persistent storage.

Since November 2021 when it was deployed on the EPN cluster and up to the writing of this article (2023-08-01), it has transferred approximately 100 million files for a total of 150 PB of experiment data. During this period, a maximum aggregated transfer speed of 42 GB/s was obtained for proton-proton data taking.

References

[1] C. Organization, *Alice detects quark-gluon plasma, a state of matter thought to have formed just after the big bang*, https://home.cern/science/experiments/alice, last accessed: 12 June 2023.

[2] M. Richter, for the ALICE Collaboration, *A design study for the upgraded ALICE O2 computing facility*, in *Journal of Physics: Conference Series* (IOP Publishing, 2015), Vol. 664, p. 082046, `https://iopscience.iop.org/article/10.1088/1742-6596/664/8/082046/pdf`

[3] C. Organization, *Eos - open storage*, `https://eos-web.web.cern.ch/eos-web/`, last accessed: June 20, 2023.

[4] P. Konopka, B. von Haller, for the ALICE Collaboration, *The ALICE O2 data quality control system*, in *EPJ Web Conf.* (EDP Sciences, 2020), Vol. 245, p. 01027, `https://inspirehep.net/files/a52ba6ddbbe69358031723a736f8df93`

[5] F. Furano, A. Hanushevsky, A. Dorigo, P. Elmer, *Xrootd - a highly scalable architecture for data access*, `https://xrootd.slac.stanford.edu/presentations/xpaper3_cut_journal.pdf`, last accessed: 12 July 2023.

[6] Q. GUI, *What is the newly added xxhash algorithm?*, `https://www.quickhash-gui.org/what-is-the-newly-added-xxhash-algorithm/`, last accessed: 12 June 2023.

[7] A.F. Suiu, M. Carabas, S. Weisz, C. Grigoras, N. Tapus, *File Spooler and Copy System for Fast Data Transfer* (ECBS 2021: 7th Conference on the Engineering of Computer Based Systems, 2021), Vol. 18, pp. 1–5, `https://doi.org/10.1145/3459960.3461559`

[8] G. Cloud, *Implementing exponential backoff*, `https://cloud.google.com/iot/docs/how-tos/exponential-backoff`, last accessed: 10 June 2023.

[9] C. Cirstoiu, C. Grigoras, L. Betev, A. Costan, I. Legrand, *Monitoring, Accounting and Automated Decision Support for the ALICE Experiment Based on the MonALISA Framework*, in *GMW '07: Proceedings of the 2007 workshop on Grid monitoring* (HPDC07: International Symposium on High Performance Distributed Computing, 2007), Vol. 245, pp. 39–44, `https://dl.acm.org/doi/pdf/10.1145/1272680.1272688`

[10] A.F. Şuiu, *Epn2eos repository*, `https://github.com/alicesuiu/FileSpooler/tree/main/src/spooler`, last accessed: 11 July 2023.