Computing and Software for Big Science (2022) 6:16
https://doi.org/10.1007/s41781-022-00089-z

ORIGINAL ARTICLE

Analyzing WLCG File Transfer Errors Through Machine Learning

An Automatic Pipeline to Support Post-mortem Distributed Data Management

Luca Clissa[1,2] · **Mario Lassnig**[3] · **Lorenzo Rinaldi**[1,2]

Received: 20 April 2022 / Accepted: 20 September 2022

Abstract
The increasingly growing scale of modern computing infrastructures solicits more ingenious and automatic solutions to their management. Our work focuses on file transfer failures within the Worldwide Large Hadron Collider Computing Grid and proposes a pipeline to support distributed data management operations by suggesting potential issues to investigate. Specifically, we adopt an unsupervised learning approach leveraging Natural Language Processing and Machine Learning tools to automatically parse error messages and group similar failures. The results are presented in the form of a summary table containing the most common textual patterns and time evolution charts. This approach has two main advantages. First, the joint elaboration of the error string and the transfer's source/destination enables more informative and compact troubleshooting, as opposed to inspecting each site and checking unique messages separately. As a by-product, this also reduces the number of errors to check by some orders of magnitude (from unique error strings to unique categories or patterns). Second, the time evolution plots allow operators to immediately filter out secondary issues (e.g. transient or in resolution) and focus on the most serious problems first (e.g. escalating failures). As a preliminary assessment, we compare our results with the Global Grid User Support ticketing system, showing that most of our suggestions are indeed real issues (direct association), while being able to cover 89% of reported incidents (inverse relationship).

Keywords Machine learning · Text processing · Clustering · Distributed data management · WLCG

Introduction

To cope with the growing amount of data to store and process [12], the big data players of both industry and academy have gradually moved to new computing paradigms in recent years. For instance, alternative infrastructures such as distributed and cloud computing have been specifically designed to address these new requirements, exploiting multiple resources geographically distributed and accessible via a network. However, the boost in performance guaranteed by these technologies comes with the price of requiring very complex interactions of both hardware and software components. Indeed, the wider the infrastructure, the higher the chances of something going wrong and the bigger the effort to detect, inspect and solve the issues. For this reason, we propose a data-driven pipeline to reduce the workload for maintaining the infrastructure integrity. Although applicable to several use cases, the presented approach is discussed in the framework of data transfer failures within the Worldwide Large Hadron Collider Computing Grid (WLCG) [7].

Background

WLCG is a global collaboration that links up more than 170 computing centers in 42 countries, serving an audience of more than 12000 physicists all around the world. The WLCG mission is to provide computing resources to store, distribute and analyze the data generated by the Large Hadron Collider (LHC). Given the scale and complexity of the LHC data, this requires massive storage facilities, immense computing power, global networking, tailored software, adequate personpower and, of course, funding. To achieve such

✉ Luca Clissa
luca.clissa2@unibo.it

[1] National Institute for Nuclear Physics, Bologna, Italy

[2] Department of Physics and Astronomy, University of Bologna, Bologna, Italy

[3] CERN, Geneva, Switzerland

Published online: 22 October 2022

challenging goals, WLCG leverages a distributed computing paradigm, where resources are shared among member states and made equally available to all partners, regardless of their physical location.

Scientific data are arguably the most valuable asset of the High-Energy Physics community. As a consequence, they are continuously transferred across the grid for several purposes, and a paramount part of the WLCG operations involves Distributed Data Management (DDM) processes. Indeed, stringent workflows are put in place by the experiments to ensure data distribution and redundancy, thus preventing data loss and guaranteeing reliable accessibility. Also, analysis workflows require individual researchers to transfer data of interest for their analyses. This potentially requires retrieving data from geographically distributed and heterogeneous storage resources (e.g. tape or disk), transferring them to computing resources that may be situated elsewhere, and transferring the results back to their machines to conduct their studies.

As a result, massive amounts of data are constantly moved across the grid thanks to various services for file transfer that perform "third party copy", i.e. a direct storage-to-storage movement of files without routing them through the client. These services are used alternately or concurrently to create a chain of software interfaces between the end-users and the physical resources. At the lowest level there is the File Transfer System (FTS) [25], which is configured to reliably interact with diverse storage devices and filesystems, execute fault-tolerant transactions and support users authentication. On top of that, the various collaborations may add other middleware layers as higher-level interfaces for the users. For example, ATLAS [5] uses an open-source framework called Rucio [6] that orchestrates the transfers, creating a catalog to track data locations, managing replication rules and retries in case of failures. Clearly, ensuring high quality of service—in terms of transfer error rates, data integrity and resources availability—is very hard due to the huge volumes transferred, the heterogeneity of the software and hardware components and the large user base.

In practice, occasional faults may happen at various levels during data transfers, which may include a wide range of root causes, provoking failures during the shipment of the files. When this happens, FTS collects the output statuses of all sub-systems that play a role during the transfer, and it concatenates them sequentially into a single message that can be inspected for debugging. These errors may vary from naive ones—e.g. a mistyped command or the request of an unavailable file—to more severe software and hardware defects. For instance, the requesting endpoint or archiving server might be temporarily unreachable (connection shortage). Likewise, the requested data may be corrupted (checksum error) due to storage hardware faults or unstable connections (network problem). Also, there might be timeouts when the shipment takes more than the pre-configured waiting window—e.g. when the desired data are bigger than usual and/or must be retrieved from tape, thus requiring more time. In addition, errors of a different nature may often arise due to the interactions between miscellaneous middleware layers. All of these factors, and more, can generate significant service disruptions and infrastructure malfunctions that require prompt intervention. For this reason, data transfer processes are continuously monitored by teams of operators. When an issue is detected, the operators report it through the Global Grid User Support (GGUS) ticketing system [3], and experts and site maintainers take care of their solution.

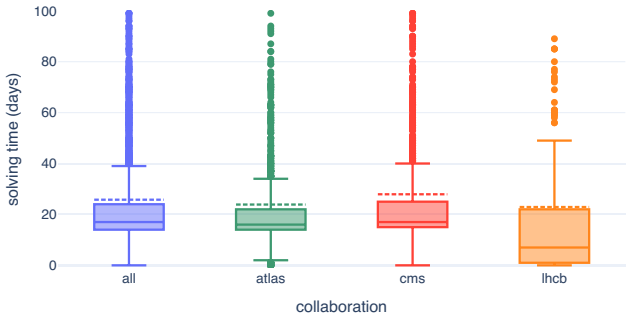
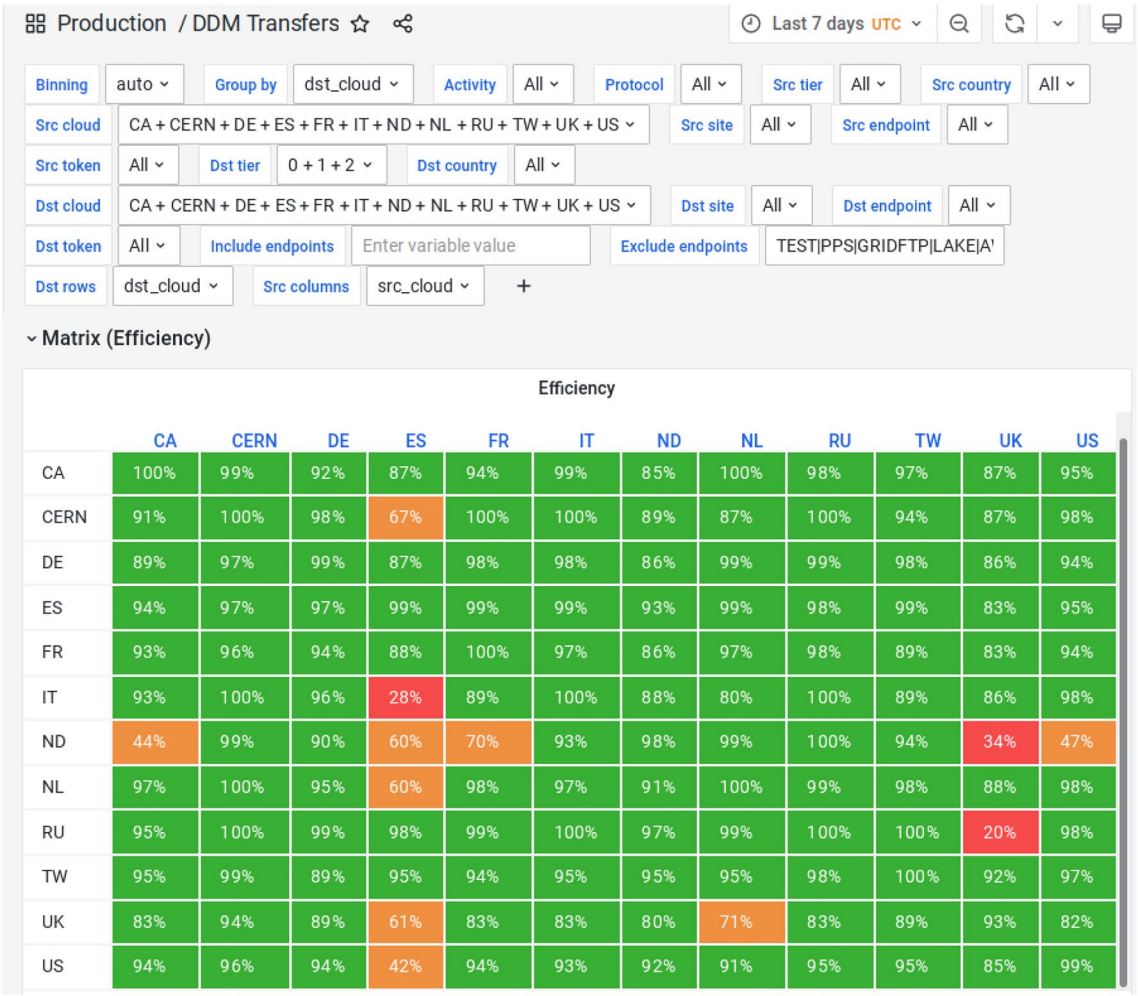


Fig. 1 Tickets solving time. Boxplot of the distribution of the solving time for GGUS incidents reported in 2019 by ATLAS, CMS and LHCb collaborations. The box sits on the central half of the distribution (25-th–75th percentiles), while the whiskers span the standard +/– 1.5 times the interquartile range. The solid and dashed horizontal lines indicate the median and the mean, respectively

To give an idea of the volumes involved, after the last LHC run[1] the ATLAS collaboration alone experienced an average traffic of more than 2 PB per day in 2019 [10], corresponding to roughly 1.5–2 million files moved each day. Nearly 10% of these transfers failed producing about 100–200k errors on a daily basis. In total, transfer failures generated more than 4k incident reports filed in 2019 for all the LHC experiments (1141 for ATLAS only). Due to the complexity of the infrastructure and its layered composition, understanding the root causes of the problem and fixing them requires time and demands a great human effort—a few dozens of part-time contributors [36]—which may entail disruptions of service. The average solving time may vary from a few hours or days—e.g. in the case of issues that are easy to solve or have already been dealt with in the past—to entire weeks—e.g. for unknown problems or more troublesome malfunctions that imply important software or hardware interventions. In practice, the median solving time for incidents reported by the ATLAS, CMS and LHCb

[1] The figures presented in this paragraph remained quite stable during the LHC shutdown and are still indicative of current loads.



	CA	CERN	DE	ES	FR	IT	ND	NL	RU	TW	UK	US
CA	100%	99%	92%	87%	94%	99%	85%	100%	98%	97%	87%	95%
CERN	91%	100%	98%	67%	100%	100%	89%	87%	100%	94%	87%	98%
DE	89%	97%	99%	87%	98%	98%	86%	99%	99%	98%	86%	94%
ES	94%	97%	97%	99%	99%	99%	93%	99%	98%	99%	83%	95%
FR	93%	96%	94%	88%	100%	97%	86%	97%	98%	89%	83%	94%
IT	93%	100%	96%	28%	89%	100%	88%	80%	100%	89%	86%	98%
ND	44%	99%	90%	60%	70%	93%	98%	99%	100%	94%	34%	47%
NL	97%	100%	95%	60%	98%	97%	91%	100%	99%	98%	88%	98%
RU	95%	100%	99%	98%	99%	100%	97%	99%	100%	100%	20%	98%
TW	95%	99%	89%	95%	94%	95%	95%	95%	98%	100%	92%	97%
UK	83%	94%	89%	61%	83%	83%	80%	71%	83%	89%	93%	82%
US	94%	96%	94%	42%	94%	93%	92%	91%	95%	95%	85%	99%

Fig. 2 Transfer efficiency matrix (Grafana). Transfer sources are shown as columns and destinations as rows. The drop-down menus at the top allow for custom filtering at the desired level of granularity

collaborations in 2019 was around 17 days, with a 90th percentile of 44 days and a long tail extending over 100 days (see Fig. 1).

When a transfer failure happens, the FTS log files are parsed and the most relevant transfer features are extracted and re-organized in a structured format. In particular, this involves collecting the exit status of each of the subsystems responsible for the transfer and appending them to compose a global error message. This information is then exposed to the on-duty operators along with other characteristics—e.g. source and destination endpoints, file size, exchange protocol and so on—and visualizations—e.g. time evolution plots or site transfer efficiency—for more in-depth investigations.

Current operations are based on a *site-centric* approach where trained personnel monitor the status of the various services almost 24/7 and try to spot hints of incorrect or undesired behaviors. In particular, the operators look at Grafana dashboards to get a high-level overview of the system. A usual starting point is the so-called efficiency matrix (Fig. 2), where the percentage of successful transfers is reported. The granularity level is customizable and it may range from global transfers between national cloud infrastructures involving more computing centers to a finer tracking of particular site exchanges or even specific endpoint links. When the efficiency falls below an acceptable threshold, typically 60–70%, on-duty operators start to investigate the issue at a lower level by checking (i) where the error happened, (ii) how many errors are produced, (iii) what is the time pattern (temporary, extended or cyclical) and (iv) which error messages are generated. However, this procedure gives rise to many false alarms as it is usual to encounter problems that do not represent a real concern. For instance, this may happen when few transfers are attempted so even a low number of errors imply a high failure rate, or when there are after-effects of a transient issue that had already been fixed. Likewise, sometimes unnecessary drill-down activity

is performed for actual issues that were already known, as in the case of ongoing tickets or site downtimes, for which reporting is not required. As a result, many human resources are employed in repetitive tasks that would enormously benefit from automation.

In addition to that, the site-centric strategy described above has some drawbacks. Firstly, monitoring focuses on spotting where issues occur, while understanding the actual root causes is typically demanded of site experts in a subsequent investigation. Secondly, problems generating few error messages are usually ignored. This is natural, and to some extent desirable, as having limited resources forces us to address bigger malfunctioning first. However, that could be a potential pitfall in cases where promptly fixing a minor issue may prevent the appearance of a more significant and longer to solve defect.

All these problems could be tackled programmatically by standardizing the logging output of all the services. In this way, neat error messages would point directly to the source of the problem, thus allowing complete automation. However, the distributed nature of the infrastructure hampers such an approach. In fact, the heterogeneous nature of WLCG computing resources and their intricate interactions demands for custom adjustments and local configurations which are just too complex to accommodate using a static strategy only. Hence, all these considerations expose the need for an intelligent support tool for speeding up infrastructure management to meet the productivity requirements for the near future.

Related Works

The automation of infrastructure management and maintenance has become crucial in recent years. The increasingly large scale of modern data centers, and the adoption of distributed resources that necessitate the interaction of diverse hardware and software components, have made this task extremely complex. Consequently, traditional approaches to infrastructure management where manual human intervention is required have become impractical or even useless. For this reason, several communities involved in the Worldwide LHC Computing Grid have started a project named Operational Intelligence[2] that aims at increasing the level of automation in computing operations, thus reducing human interventions. As a result of the joint effort, several strategies have already been proposed to support operational workflows in various ways [16–18, 26]. Some works address anomaly detection by leveraging overall workloads—e.g. number of running processes, hardware resources usage, network saturation—as indicators of infrastructure health and monitoring their trends over time.

The deviations from normal operations are considered anomalies and trigger alerts to be investigated by experts [21].

Other attempts rely on event logs as the primary way to register key runtime information. These reports record events happening during the execution of a system to provide an audit trail that can be helpful to understand the system activity and diagnose problems. This information can be exploited in various forms. Some approaches focus on log activity summary statistics (e.g. number of printed lines) and try to disentangle nominal behaviors from suspect activity [13, 14, 31]. Other alternatives use the log content instead, thus directly analyzing the textual information contained in the log files [20]. These vary from traditional keyword searches—e.g. “kill”, “error”, “fail”, “exception”— and heuristics [37] to smarter tools based on deep learning language models.

Another interesting approach suggests using a convenient pipeline to group logs of failed jobs and exploit the knowledge coming from previous failures [27]. After substituting placeholders instead of parametric parts, the textual information of each log line is encoded (vectorization stage) based on Inverse-Document event Frequency (IDF) and contrast-based weighting. The resulting numerical representation undergoes an agglomerative hierarchical clustering algorithm that finds groups of similar logs. The resulting clusters are then summarized by their centroids and compared to a knowledge base of previous failures and corresponding solutions. If the sequence similarity to one of the known issues is above a given threshold, the corresponding actions are applied to solve the problem. Otherwise, the log sequence is passed to system experts for manual inspection and the reference dataset is successively updated. In this way, human resources are involved only in handling new issues, while previous knowledge is exploited for recurrent ones.

Some approaches specifically target data processing workflows within WLCG, with a focus on error messages coming from failed analysis jobs [22]. First, the error messages are tokenized (for more details see Sect. 2.2) and cleaned from digits, punctuation and special characters. Then, a hashing algorithm replaces the parametric parts of the message with a placeholder, and the resulting patterns are exploited for the following elaborations. In this way, the total amount of data is reduced by 90–95%. After the above pre-processing, the vectorization stage is based on word-2vec [30] that computes a numerical representation for each token. The overall message representation is then retrieved by averaging over single word embeddings. The resulting representation is then reduced in dimension by means of principal components analysis [39], and a DBSCAN [19] algorithm is adopted for the clustering stage. Finally, cluster

[2] For more details: https://operational-intelligence.web.cern.ch/.

descriptions are extracted by searching common textual patterns and key phrases for all messages belonging to the same cluster.

The advantage of text-based pipelines is that the textual information can aid system experts finding root causes and explanations which are harder to grasp from the amount of logging activity alone. However, the above methods require significant pre-processing that may need deep customization for specific data, which hampers their adaptation to novel use cases with possibly diverse logging conventions, terminology and structure. Furthermore, no additional information (e.g. site, time pattern) is leveraged apart from the text itself, which limits their practical impact.

Contribution

The goal of this work is to discuss a complementary approach to support current DDM operations for grid monitoring based on a computer-aided strategy independent of experiment-specific settings. In particular, we propose a pipeline that takes into account FTS error messages, source and destination hostnames, and time patterns. *Unsupervised machine learning* techniques are then leveraged to identify clusters of similar failures that act as suggestions of potential issues for on-duty operators. Also, we perform a post-mortem analysis to test our approach in a real-world scenario, showing that: (i) our approach is able to find groups of similar errors and (ii) the proposed visualization enables to spot quickly what failures are more frequent, where they occur and whether their time trend is of concern. Furthermore, we compare our results with service tickets and show how the highlighted clusters reflect the issues reported by the operators. Finally, we provide a full, scalable implementation[3] developed in compliance with the Operational Intelligence software framework[4] to allow fast integration and testing by the whole LHC community.

Methods

The pipeline proposed in this work comprises an initial pre-processing step followed by the *vectorization*, *clustering* and *description stages*. Figure 3 reports a diagram that summarizes our workflow from the initial error message to the final outputs, and the next subsections provide a thorough description of each of the stages.

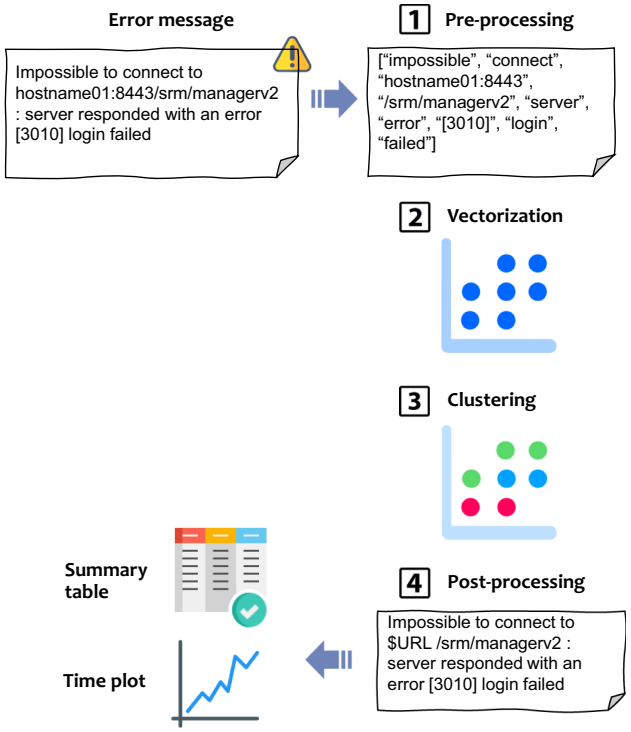


Fig. 3 Pipeline diagram. The error message is first pre-processed and split into tokens (1). Then, the vectorization stage transforms the textual information into numeric data (2). The next step is clustering, where similar error messages are grouped (3). Finally, the messages are post-processed to get common patterns (4) and the resulting clusters are presented to the operators in the form of a summary table and time evolution plots

Pre-Processing

Our approach applies minimal pre-processing to limit hard-coded feature engineering and let the subsequent vectorization stage figure out linguistic features of the error messages—e.g. grammar, syntax, lexicon and semantic—on its own. The rationale behind this choice is that the resulting representation should be more expressive, thus better modeling the semantic of the messages and easing the successive clustering phase.

With this goal in mind, the raw error strings are first transformed to lowercase and enriched by appending the source and destination hostnames. In particular, both hostnames are inserted at the end of each message with prepended `src_` or `dst_` prefixes to distinguish whether they were involved as source or destination, respectively. The resulting text then undergoes a process of quantization whereby the raw strings are decomposed into unitary pieces of information. This process is commonly referred to as **tokenization** and the resulting atomic units are called tokens. In our case, we resort to whitespace tokenization for the sake of simplicity, which means individual words are

[3] https://l.infn.it/opint-pyspark.

[4] https://l.infn.it/opint-framework.

Table 1 Message pre-processing pipeline

Raw message	"DESTINATION OVERWRITE srm-ifce err: Communication error on send, err: [SE][srmRm][] httpg://hostname01.Site-4.ch:8443/srm/managerv2: CGSI-gSOAP running on fts-address-004.cern.ch reports Error initializing context GSS Major Status: Authentication Failed GSS Minor Status Error Chain: globus_gsi_gssapi: SSL handshake problems globus_gsi_callback_module: Could not verify credential globus_gsi_callback_module: Could not verify credential globus_gsi_callback_module: The certificate has been revoked: Serial number = -1 (0xFFFFFFFFFF"
Append hostnames	"DESTINATION OVERWRITE srm-ifce err: Communication error on send, err: [SE][srmRm][] httpg://hostname01.Site-4.ch:8443/srm/managerv2: CGSI-gSOAP running on fts-address-004.cern.ch reports Error initializing context GSS Major Status: Authentication Failed GSS Minor Status Error Chain: globus_gsi_gssapi: SSL handshake problems globus_gsi_callback_module: Could not verify credential globus_gsi_callback_module: Could not verify credential globus_gsi_callback_module: The certificate has been revoked: Serial number = -1 (0xFFFFFFFFFF src_srmatlas.pic.es dst_hostname01.Site-4.ch"
Tokenization	["destination", "overwrite", "srm-ifce", "err:", "communication", "error", "on", "send,", "err:", "[se][srmrm][]", "httpg://hostname01.Site-4.ch:8443:/srm/managerv2:", "gsi-gsoap", "running", "on", "fts-atlas-005.cern.ch", "reports", "error", "initializing", "context", "gss", "major", "status:", "authentication", "failed", "gss", "minor", "status", "error", "chain:", "globus_gsi_gssapi:", "ssl", "handshake", "problems", "globus_gsi_callback_module:", "could", "not", "verify", "credential", "globus_gsi_callback_module:", "could", "not", "verify", "credential", "globus_gsi_callback_module:", "the", "certificate", "has", "been", "revoked:", "serial", "number", "=", "-1", "(0xfffffffffff", "src_srmatlas.pic.es", "dst_hostname01.Site-4.ch"]
Remove punctuation	["destination", "overwrite", "srm-ifce", "err", "communication", "error", "on", "send", "err", "[se][srmrm][]", "httpg://hostname01.Site-4.ch:8443:/srm/managerv2", "cgsi-gsoap", "running", "on", "fts-atlas-005.cern.ch", "reports", "error", "initializing", "context", "gss", "major", "status", "authentication", "failed", "gss", "minor", "status", "error", "chain", "globus_gsi_gssapi", "ssl", "handshake", "problems", "globus_gsi_callback_module", "could", "not", "verify", "credential", "globus_gsi_callback_module", "could", "not", "verify", "credential", "globus_gsi_callback_module", "the", "certificate", "has", "been", "revoked", "serial", "number", "=", "1", "(0xfffffffffff", "src_srmatlas.pic.es", "dst_hostname01.Site-4.ch"]
Remove stopwords	["destination", "overwrite", "srm-ifce", "err", "communication", "error", "send", "err", "[se][srmrm][]", "httpg://hostname01.Site-4.ch:8443:/srm/managerv2,cgsi-gsoap", "running", "fts-atlas-005.cern.ch", "reports", "error", "initializing", "context", "gss", "major", "status", "authentication", "failed", "gss", "minor", "status", "error", "chain", "globus_gsi_gssapi", "ssl", "handshake", "problems", "globus_gsi_callback_module", "verify", "credential", "globus_gsi_callback_module", "verify", "credential", "globus_gsi_callback_module", "certificate", "revoked", "serial", "number", "=", "1", "(0xfffffffffff", "src_srmatlas.pic.es", "dst_hostname01.Site-4.ch"]
Url split	["destination", "overwrite", "srm-ifce", "err", "communication", "error", "send", "err", "[se][srmrm][]", "httpg://hostname01.Site-4.ch:8443", "/srm/managerv2", "cgsi-gsoap", "running", "fts-atlas-005.cern.ch", "reports", "error", "initializing", "context", "gss", "major", "status", "authentication", "failed", "gss", "minor", "status", "error", "chain", "globus_gsi_gssapi", "ssl", "handshake", "problems", "globus_gsi_callback_module", "verify", "credential", "globus_gsi_callback_module", "verify", "credential", "globus_gsi_callback_module", "certificate", "revoked", "serial", "number", "=", "1", "(0xfffffffffff", "src_srmatlas.pic.es", "dst_hostname01.Site-4.ch"]

The table illustrates the pre-processing steps (left) and the resulting data (right) for a sample error message. The raw error string is reported at the top, and the resulting pre-processed data at the bottom

used as tokens. Once tokens are obtained, they are stripped of leading and trailing punctuation (`":;,.-"`) . After that, tokens corresponding to common English stopwords (refer to `pyspark.ml.feature.StopWordsRemover` documentation[5] for a full list) or unuseful punctuation (`":-+"`) are discarded. Finally, the URL addresses are split into two components: the net location and the relative path of the requested resources. For instance, `httpg://::/srm/managerv2` is decomposed as `httpg://:` and `srm/managerv2` . In this way, it is possible to exploit the compositional structure of the URL addresses to reduce the vocabulary of unique tokens. Also, this allows the model to disentangle the contribution of the single parts in different messages. The above transformations are illustrated for a sample error message in Table 1.

Vectorization

The vectorization stage transforms the pre-processed text of each error message into numeric information that quantitative techniques can digest. Ideally, we would like to map each message to a point in a vectorial subspace (*embedding*) where "similar" messages are close to each other, so that they can be subsequently grouped based on their location. Although more recent and powerful alternatives are available for this purpose [9, 15, 34], they do not work well with short-text data [1]. Thus, we adopt the simpler yet effective **word2vec** language model [30] (skip-gram architecture

[5] https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.StopWordsRemover.html.

implemented by `pyspark`) to compute message embeddings starting from pre-processed tokens. Specifically, the model is trained once on one month of data—from 2020-10-01 to 2020-10-31—for a total of nearly 28.6 M error messages, corresponding to a vocabulary of 970 unique tokens. The pre-trained model is then re-used as-is for online application, possibly updating it once in a while—say every 4–6 months. The assumption underneath this strategy is that semantic and syntactic relationships within messages should be stable or slowly-varying across time, thus not requiring frequent updates of token's embedded representations. Once the model is trained, the resulting word embeddings are used to transform single tokens into numerical vectors, and they are then averaged to get the corresponding message representations. Regarding hyper-parameters, the window size, the embedding size and the minimum count were the ones affecting the final representation the most. For this reason, a grid search is conducted to compare alternative parametrizations in terms of the following clustering performance. In practice, the word2vec model is trained with a given hyper-parameters configuration and then used as embedding for the subsequent clustering stage. The latter is repeated several times for errors happening on different days (not included in the word2vec training period), and the optimal configuration is then chosen based on the compactness of the resulting groups in terms of the WSSE and ASW metrics introduced in Sect. 2.3. The values of $w = 12$ (window size), $h = 300$ (embedding size) and $min\_count = 50$ (minimum count) seem to work best in our experiments and are therefore adopted in the following.

Clustering

The next step of the pipeline is the clustering stage. In this study, we resort to clustering for grouping messages including similar content. The resulting clusters are therefore interpreted as error categories. The idea is to repeat this stage for online processing of new data bunches, e.g. every day, every shift or every 4 hours. In fact, conversely to the message structure, the malfunctions observed in the infrastructure may change from day to day, which calls for a more flexible definition of what and how many error categories are present in the analyzed data.

In practice, we adopt a slight variation of the k-means algorithm [28] referred to as **k-means++** [4]. Although more advanced clustering algorithms are available and may be applied to our use case [2, 19, 29, 33, 41], the choice of a k-means algorithm is justified by its intuitive approach and good performance in practice in a wide range of applications [38]. Also, a perhaps more profound and substantial motive is that the clustering strategy may be seen as a functional but not primary pipeline stage. Indeed, the learned language model determines the geometry of the embedded space, thus influencing the point cloud shapes of different error categories. For this reason, we embrace the idea that a simple clustering algorithm is preferable, and particular attention must be devoted to tuning the vectorization stage for easing the subsequent clustering, possibly even fostering the learning of an optimal representation for a specific clustering algorithm [40].

To demonstrate the approach, we report the analysis of FTS data from one full day of operation (2021-01-15)—corresponding to roughly 1 M errors and 1.5 GB of data—, where the *cosine similarity* is adopted as common practice in similar applications. To help the successive evaluation phase, only transfers between pledged WLCG resources (namely Tier-0, Tier-1s and Tier-2s) are considered in the analysis, thus discarding transfers involving local clusters or HPC centers.

The number of clusters, k—in our case, the unknown number of error categories—is selected at each clustering stage based on a grid search for $k \in [12, 15, 20, 30]$. For this purpose, two geometrical criteria are considered to compare results of different settings. The first is the Within cluster Sum of Squared Errors (WSSE):

$$\mathrm{WSSE}(\mathrm{dist}, k) = \sum_{j=1}^{k} \sum_{x_i \in C_j} \mathrm{dist}\left(x_i - \bar{x}_j\right), \tag{1}$$

where x_i is a generic data point, *dist* is a desired distance measure, and C_j and $\bar{x}_j$ indicate a generic cluster and its centroid, respectively. The second is the Average Silhouette Width (ASW) [35]:

$$\mathrm{ASW}(\mathrm{dist}, k) = \frac{1}{n} \sum_{i=1}^{n} \frac{b_i - \bar{a}_i}{\max\left(\bar{a}_i, b_i\right)}, \tag{2}$$

where n is the total number of observations, i.e. error messages in our case, $\bar{a}_i$ is the average distance of x_i from all the other points belonging to the same cluster C_I, and b_i is the minimum average distance of x_i from the observations in all the other clusters $C_j, \forall j \neq I$. The WSSE measures the internal cluster variability, so the lower its value, the better the performance. The ASW, instead, accounts for both internal homogeneity and external separation of the clusters (bounded in the interval $[-1, +1]$: the closer to 1, the better). Given the more intuitive reading of ASW values, the latter is used in the following as the main figure of merit. The results of the comparison between WSSE and ASW for different values of k are reported in Fig. 4. Both indicators tend to improve as the number of clusters increases. In particular, a value of $k = 30$ clusters seems to be optimal according to both criteria. Notably, however, the ASW indicator reaches very high values (around 0.9) even for lower k values, which means nearly-optimal performances can be achieved with fewer clusters. For this reason, the configuration having $k =$

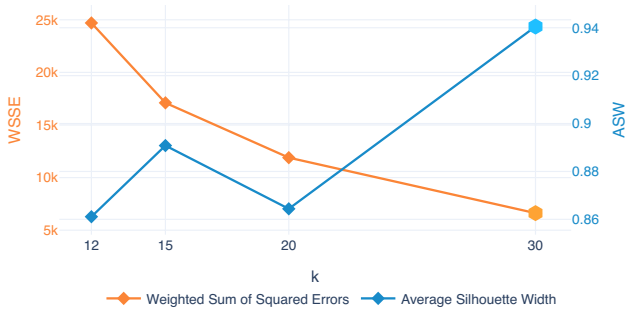


Fig. 4 Optimization of k. The plot shows the value of the WSSE and ASW metrics as a function of the number of clusters, k. The hexagonal markers indicate the optimal values, which correspond to $k = 30$ for both indicators

15 is preferred to limit the number of suggested issues and minimize the operators effort.

Cluster Description

The last stage of the pipeline is the cluster description. This step is fundamental to present the results in the most intelligible and immediate format for end-users. Indeed, given the unsupervised learning approach adopted, the interpretation of the clustering output resorts to the manual inspection of each group's content. This, in turn, potentially means reading hundreds of error strings, comparing the source and destination information, and spotting suspect time patterns. Therefore, producing a nice and compact visualization of the results is paramount to make the approach effective and avoid excessive manual checks by the operators. For this reason, the clustering results are summarized into two complementary outputs that are presented to the operators.

First, the **summary table** Figs. 5 and 6 represents the most important and informative visualization. This output is obtained by a first pre-aggregation of the clusters and is organized in a tabular format. The first three columns provide numeric summaries concerning the cluster size, the number of unique strings within each group, and the corresponding number of unique patterns. The latter is

ID	cluster size	# strings	# patterns	Top 3				
				message	n	%	source rcsite	destination rcsite
0	819465	117	14	destination overwrite srm-ifce err communication error on send err [se][srmrm][] $URL /srm/managerv2 cgsi-gsoap running on $ADDRESS reports error initializing context gss major status authentication failed gss minor status error chain globus_gsi_gssapi ssl handshake problems globus_gsi_callback_module could not verify credential globus_gsi_callback_module could not verify credential globus_gsi_callback_module the certificate has been revoked serial number = 1 (0xffffffffff	85545	10.44%	Site-1	Site-4
				destination overwrite srm-ifce err communication error on send err [se][srmrm][] $URL /srm/managerv2 cgsi-gsoap running on $ADDRESS reports error initializing context gss major status authentication failed gss minor status error chain globus_gsi_gssapi ssl handshake problems globus_gsi_callback_module could not verify credential globus_gsi_callback_module could not verify credential globus_gsi_callback_module the certificate has been revoked serial number = 1 (0xffffffffff	84453	10.31%	Site-2	Site-4
				destination overwrite srm-ifce err communication error on send err [se][srmrm][] $URL /srm/managerv2 cgsi-gsoap running on $ADDRESS reports error initializing context gss major status authentication failed gss minor status error chain globus_gsi_gssapi ssl handshake problems globus_gsi_callback_module could not verify credential globus_gsi_callback_module could not verify credential globus_gsi_callback_module the certificate has been revoked serial number = 1 (0xffffffffff	77410	9.45%	Site-3	Site-4
6	9673	347	60	source srm_get_turl srm-ifce err connection timed out err [se][statusofgetrequest] [etimedout] \$URL /srm/managerv2 user timeout over	1838	19.00%	Site-22	Site-46
				transfer globus_ftp_client the server responded with an error 421 service busy connection limit exceeded please try again later closing control connection	522	5.40%	Site-33	Site-47
				transfer globus_ftp_client the server responded with an error 421 service busy connection limit exceeded please try again later closing control connection	300	3.10%	Site-29	Site-47
3	34183	1568	1537	error reported from srm_ifce 2 [se][ls][srm_invalid_path] no such file or directory	13118	38.38%	Site-12	Site-35
				error reported from srm_ifce 2 [se][ls][srm_invalid_path] no such file or directory	9333	27.30%	Site-12	Site-17
				error reported from srm_ifce 2 [se][ls][srm_invalid_path] no such file or directory	1707	4.99%	Site-12	Site-22

Fig. 5 Summary table: successes. The figure illustrates the main achievements of the pipeline. Cluster 3 provides immediately clear indication of the error type, i.e. `message`, and where it occurs (green). The others also suggest the approach is actually learning to understand message parameters and message semantic (yellow, clusters 0 and 6)

ID	cluster size	# strings	# patterns	Top 3				
				message	n	%	source rcsite	destination rcsite
4	51370	10108	814	transfer globus_ftp_client the server responded with an error 500 command failed open/create [error] server responded with an error [3021] unable to get quota space quota not defined or exhausted $FILE_PATH disk quota exceeded	4912	9.56%	Site-7	Site-31
				transfer globus_ftp_client the server responded with an error 500 command failed open/create [error] server responded with an error [3021] unable to get quota space quota not defined or exhausted $FILE_PATH disk quota exceeded	3709	7.22%	Site-8	Site-31
				error on $IPv6 [error] server responded with an error [3010] login failed	2950	5.74%	Site-9	Site-33
2	15132	11	9	transfer globus_ftp_control gss_init_sec_context failed globus_gsi_callback_module could not verify credential globus_gsi_callback_module could not verify credential globus_gsi_callback_module the certificate has been revoked serial number = 1 (0xffffffffffffffff) subject=/c=bm/o=quovadis limited/cn=quovadis grid ica g2	2048	13.53%	Site-27	Site-42
				destination srm_put_turl error on the turl request [se][statusofputrequest] [srm_duplication_error] cannot srmput file because it already exists!	1431	9.46%	Site-28	Site-12
				destination srm_put_turl error on the turl request [se][statusofputrequest] [srm_duplication_error] cannot srmput file because it already exists!	914	6.04%	Site-15	Site-12

Fig. 6 Summary table: limitations. The two clusters show evidence of contamination (red) due to generic partial matching (yellow, cluster 4) or outliers aggregation (cluster 2)

obtained from the raw strings by means of an *abstraction mechanism*[6] that removes the parametric parts—like file paths, IP addresses, URLs, checksum values, and so on—and replaces them by parameter-specific placeholders—e.g. `$FILE_PATH`, `$ADDRESS`, `$URL` and `$CHECKSUM`, respectively. The core part of this visualization is then represented by the *Top 3* section. Here, the three most frequent triplets of `<pattern>-<source>-<destination>` are reported in descending order for each cluster, alongside their cardinality and the percentage over the cluster size. Such information provides several precious insights for spotting the source of potential problems, e.g. whether a pattern is responsible for a large number of failures or if it accounts for a conspicuous fraction of the cluster.

In addition, this representation allows us to investigate the contribution of source/destination pairs to each cluster. In this way, it is possible to discriminate failures based on both the nature of the problem and the location where they occurred.

The second output of the pipeline consists of a **time evolution plot** depicting the temporal trend of the number of errors generated by each cluster (Fig. 7).

This piece of information is crucial to discriminate between serious issues that require immediate actions (e.g. escalating or cyclical failures, see Figs. 7a and d) and problems that are transient (Fig. 7b) or in resolution (Fig. 7c).

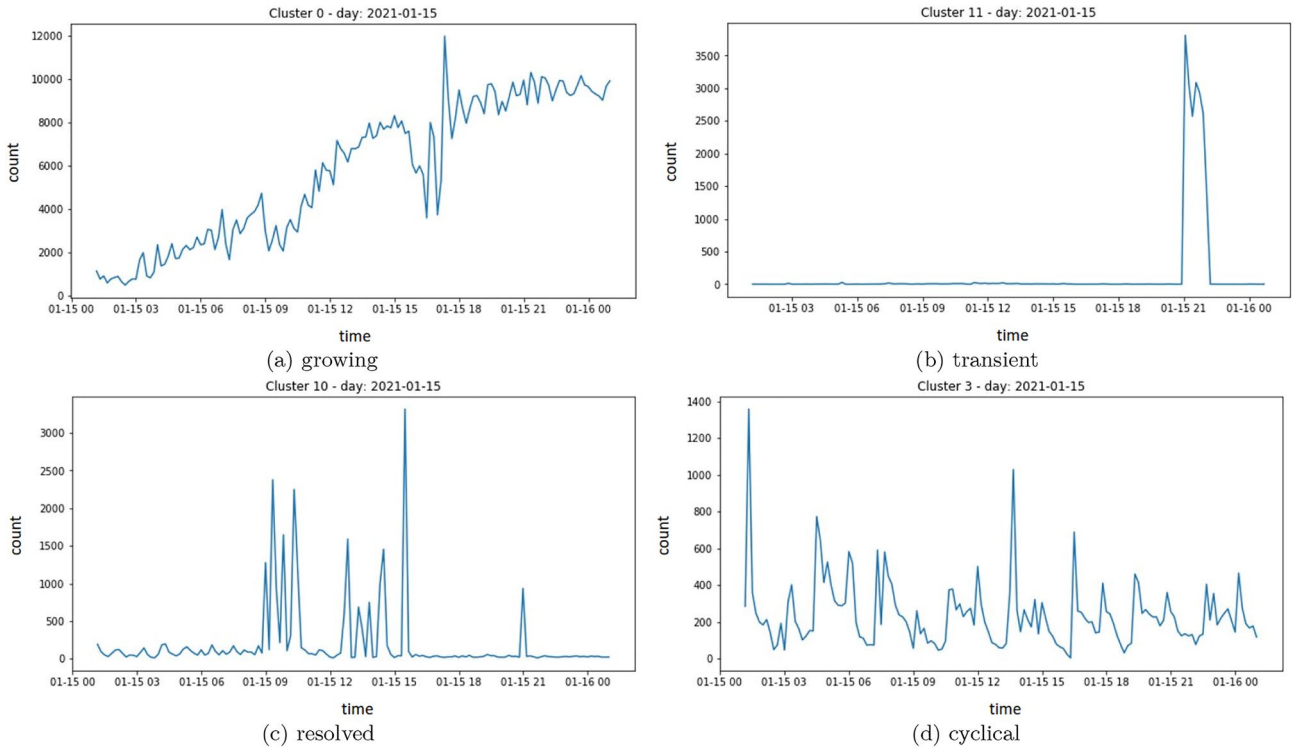
Overall, the idea behind our pipeline is to exploit the summary tables and the time plots for each cluster as suggestions of potential issues to investigate further. In this way, the operators can have a first grasp of what kind of failures are observed and their corresponding amounts (Top-3 section), also having an indication of where they are happening (source/destination sites). Moreover, by looking at the time charts it is possible to immediately discard transient (Fig. 7b) or resolved (Fig. 7c) problems based on the evolution of the number of generated failures over time.

Results

Performance assessment is one of the trickiest parts when coming to unsupervised methods [24, 38]. Although similarity metrics as ASW and WSSE can be exploited for this purpose, they only measure the internal homogeneity of clusters, and their separation from one another. Thus, these are mere geometric indicators that do not take into account the actual meaning of the data points clustered in the same group, i.e. whether the messages share similar content and/or meaning.

In light of the above concerns, our work splits the evaluation of performances into two complementary phases. First, a qualitative assessment explores the cluster contents and

[6] Refer to the full implementation for more details: https://l.infn.it/opint-abstraction.



(a) growing (b) transient (c) resolved (d) cyclical

Fig. 7 Time evolution charts. The figure illustrates several time patterns for the generated failures in 4 different clusters. Each plot reports the count of errors in bins of 10 min

expresses the goodness of fit based on their interpretability, i.e. how messages of the same cluster resemble each other's meaning. Second, a quantitative evaluation is addressed by cross-checking the clustering result against the GGUS reported incidents. In this way, a more direct measure of impact is given by reckoning the ability of our approach to mimic current operations.

Qualitative Assessment: *Interpretability*

This section presents a qualitative assessment of the clustering performance based on the interpretability of the discovered groups of messages. In particular, the discussion is articulated by simulating the operator's perspective when reading the pipeline outputs. In the following, we report five cherry-picked examples to showcase our approach's major successes and failures, articulating the discussion from the operator's perspective when reading the pipeline outputs. Specifically, we first illustrate a thorough examination of the biggest cluster discovered (see Fig. 5, cluster with id = 0).

Then we highlight some strengths and limitations of our approach, bringing other exemplary cases as evidence. The same procedure and similar conclusions apply likewise to most groups. Thus, a complete dissertation is omitted here for conciseness[7].

[7] Full results available at: https://l.infn.it/opint-results.

The main output of our pipeline is the summary table illustrated in Figs. 5, 6, which reports a succinct highlight of the cluster contents and represents the most substantial source of information. A reasonable reading approach is to start with the groups including more errors and gradually proceed with the smaller ones.

In this case, the biggest cluster is shown in Fig. 5 in the first row with id = 0. Despite including almost 820k error strings (`# cluster size`), the actual number of different messages is only 117 (`# strings`). This number further reduces to simply 14 unique patterns (`# patterns`) after the abstraction mechanism described in Sect. 2.4 is applied, which is way more manageable for manual inspection than the initial cluster size. A second insight is then provided by the Top-3 section. Including the auxiliary information about the source and destination sites involved makes it evident as the failures are united by the same error template and destination site. This suggests that `Site-4` may have a problem and that its root cause is linked to the error pattern reported in the `message` column. Finally, the last piece of information to consider is the time evolution plot (see Fig. 7a). In this case, the cluster shows an increasing trend throughout the whole day of analysis. Specifically, the number of generated failures grows from less than 2000 errors at the beginning of the day to a value around five times higher, with an increment boost from 9 a.m. onwards. By and large,

Table 2 GGUS pre-validation

N. clusters	ASW	WSSE	Perfect match	Fuzzy match	Partial match	False positives	False negatives
15	0.89	17107	7	3	2	3	1

Summary of the cross-check between clusters and incidents reported in GGUS. Most of the groups discovered are linked to reported issues, with only 3 false positives and 1 false negative

all these factors clearly advise that a potential issue is happening at `Site-4` as it always appears as a destination. Also, the message information further suggests that the failure is linked to a *revoked certificate* that cannot be verified. Finally, the time chart shows that the problem is escalating and needs prompt intervention.

Despite providing only good proxies of the actual end goals—i.e. root causes and solving actions—, this rapid analysis already points to actionable insights regarding *where* and *what* faults occur and whether they represent a real concern. Notice that one can draw similar conclusions by looking separately at the site transfer efficiency and the most frequent unique strings or patterns. However, observing high failure rates for `Site-4` only answers to *where* the faults occur. Likewise, the information contained in the errors only relates to the *what* part of the question. Thus, both approaches would lead to partial conclusions and require additional investigations to reach the same result.

Conversely, our approach addresses the two tasks together, thus letting the conclusion emerge rapidly and naturally. A further advantage is that one can leverage both site and pattern information for more precise indications. For instance, one could hypothesize that not only is `Site-4` experiencing a problem, but the issue is limited to incoming connections. Indeed, `Site-4` is involved only as a destination, and the error patterns point to something related to `destination overwrite`. Therefore, the previous advantages show how shifting from the current site-centric focus to a hybrid strategy based on error messages and auxiliary information is beneficial.

In addition to the practical usage of our pipeline, the results illustrated in Figs. 5 and 6 expose interesting insights about what the models are actually learning. For instance, the substantial reduction observed passing from errors to patterns suggests that the pipeline has learned something similar to an abstraction mechanism. Indeed, the raw error strings of `cluster 0` differ only by the `$URL` and `$ADDRESS` parameters (see `message` column). Although one may argue that the same could be obtained using a flexible parsing strategy, the superiority of our approach is even more evident in `cluster 6` (Fig. 5). In this case, the clustering joins two patterns with a far less straightforward linkage. In fact, this result appears to resemble the human association that `connection timed out` (first pattern) may be linked to a `service busy connection limit exceeded` (second and third) problem. Notably, this is a much higher level of abstraction with respect to a smart parsing approach, and it goes way beyond what one could achieve based on good abstraction heuristics. Clearly, this property is highly desirable in practice, as it testifies that the approach produces a good embedded representation and recognizes the similarity of messages sharing similar content. In particular, this holds not only up to some parametric parts but also in terms of their actual meaning. In turn, this observation corroborates the initial design choice of applying minimal pre-processing and letting the model learn by itself.

Another clear example of success is provided by the `cluster 3` (Fig. 5), where the visualization makes it immediate for the operator to understand that the issue is related to a missing file (`no such file or directory`) at `Site-12`.

However, our pipeline comes also with some limitations (Fig. 6). For instance, the two patterns reported in `cluster 4` show a more vague connection that would require more in-depth investigation. As a matter of fact, they seem to be linked due to a generic `server responded with an error` which is a very generic incipit to several error strings. Apart from that, the error codes are different (`[3021]` vs `[3010]`, which may imply the clustering is too coarse and a more refined distinction is needed. Also, the messages point to seemingly extraneous issues (storage vs authentication). Such observations expose two limitations. On the one hand, tuning the pipeline to meet the desired level of granularity when separating different groups is extremely complex. On the other hand, this behavior may be due to the difficulty in comparing longer strings (first and second patterns) with short-text (third).

Another drawback is related to how outliers are handled. The k-means algorithm is bounded to the specified number of clusters, k, which sets the number of output groups irrespectively of the underlying structure of the data. As a result, the outliers are often incorporated into the closer cluster. When the latter is big enough, they probably pass undetected as they are dispersed into a heap of other messages. However, they may contaminate other clusters when the affected group has a comparable size, as in the case of second and third patterns in `cluster 2`.

Quantitative Assessment: *GGUS Tickets*

The drawback of unsupervised techniques lies in the inherent difficulty of the evaluation phase, as no ground truth is available for comparison [38]. A first assessment may come from the similarity measures computed on the resulting clusters (see Table 2). The ASW value is around 0.9, which is close to the optimal value of 1 for such indicator. Hence the clustering stage does a decent job in discovering groups that are internally compact and well separated. However, this metric does not measure directly the content similarity between messages. In fact, the ASW treats the strings as points disentangled from their meaning, and we would have the same score by randomly shuffling the text associated to points in the embedded representation. This means that the ASW makes sense only if the word2vec embedding is appropriate, which is difficult to check. To overcome this limitation, we have conducted extensive testing using incidents reported in GGUS as a benchmark. In this way, we attempt to provide a quantitative assessment of the pipeline performances and a more direct measure of its potential impact when applied in practice. In particular, we explore the overlapping between discovered clusters and the reported issues in two directions expressing alternative perspectives to the problem. On one side, we evaluate the usefulness of our approach for the operators, i.e. how clusters explain failures/tickets (*direct association*). On the other, we study the overall capacity of the pipeline to discover and highlight issues—i.e. how many failures/tickets are reflected in the clusters (*inverse association*). In the first case, the objective is to limit the effort of the operators by suggesting as few potential failures as possible, meanwhile still highlighting the major concerns for the infrastructure. Thus, the focus is on limiting false positives at the expense of neglecting minor issues. On the contrary, the second point of view requires a more comprehensive search aimed at isolating all the ongoing malfunctions, irrespectively of their current priority. Hence, this time the focus is on maximizing true positives. Table 2 reports a summary of the evaluation according to both perspectives.

Concerning the first angle, we consider GGUS issues reported in a skewed time window of 17 days (01-01 to 01-18) around the day of the analysis for a total of 20 tickets related to data transfer failures. Adopting this filtering strategy is convenient since it considers both previously known issues and delayed detections. The former is necessary because standard practice in current operations requests not to open new incident reports when related investigations are already ongoing. Hence, considering only tickets opened on the analysis day may lead to incorrect conclusions. Instead, the latter is convenient to account for a "grace period" if the operators do not promptly spot failures that are really happening during the analysis. Overall, a good level of agreement is observed between the 15 discovered clusters and the 20 tickets. Specifically, the 7 *perfect matches* indicate cases whereby the reported message and the affected site coincide with the ones highlighted by the clusters. The 3 *fuzzy matches*, instead, refer to occasions whereby the agreement is less obvious, meaning that the cluster has evident connections with more than one ticket. Similarly, the 2 *partial matches* describe cases whereby either the message or the site coincide. The previous three statistics reveal that 12 out of the 15 suggested failures have led to fruitful investigations, thus implying a precision between 0.46 and 0.8 depending on the degree of nuisance one is willing to tolerate. Besides the above matches, 3 clusters highlight issues not reported on GGUS in the considered time window. These false positives indeed entail a futile effort for the operators and should be avoided, e.g. thwarting in-depth investigations if the temporal pattern is not escalating and/or the number of errors is not a concern. Nevertheless, in our case, posterior checks on the 3 false positives showed hints for real problems that went undetected or unreported by the operators, i.e. the error pattern seemed similar to other incidents opened to different sites.

For the second assessment, we investigate the relationship between clusters and tickets in the opposite direction, i.e. by looking at how many reported issues our approach captures. In this case, we consider a different baseline that provides a fairer detection performance evaluation. Indeed, it is reasonable to think that the failures observed during the analysis may be correlated to earlier tickets, thus justifying the adoption of a wide time window for the direct association.

However, the same rationale does not necessarily apply when we reverse our perspective. In fact, there is no prior guarantee that a past ticket will generate new failures at a given moment in the future. Hence, considering all tickets undergoing investigations would potentially bias our measurement since specific past failures may not produce new malfunctions during the day of the analysis, thus resulting in untruthful false negatives. For this reason, in the case of the inverse association we limit our baseline to consider solely the tickets for which failures were really observed during the day of the analysis, thus reducing the initial 20 reports to only 9. Given this reference framework, the clusters successfully identify 8 out of 9 tickets, thus overlooking only a single issue.

To summarize, the previous results show that the approach presents promising perspectives given the complexity of the task and the completely unsupervised approach embraced. Although conducting an indisputable quantitative assessment is challenging—if not impossible with the available data—, the considerations expressed above furnish a reasonable proxy of the potential of our approach. Of course, a trade-off between the two perspectives is desirable

in practice, for which more tuning is necessary with the help of operators and site experts.

Discussion

The huge scale of modern computing infrastructures has made automatic management solutions essential for a proper exploitation of such resources. This is particularly true for WLCG and the LHC experiments, whereby the upcoming upgrade will deliver ten times the current volumes at a flat budget for infrastructure management.

This work proposes a pipeline to support DDM operations by suggesting potential transfer failures to investigate more in depth. The approach has already undergone some pre-production integration and testing. In particular, the implementation is already compatible—at least to some extent—with the production systems as it natively interacts with the raw data streams, and it complies with the timely execution requirements for online processing. In fact, the pipeline takes around 2.5/3 h for one day of data, which is compatible with one or two applications per 8-h shifts. This runtime is almost equally divided among the clustering stage—with a grid search for the optimization of k as described in Sect. 2.3—and the post-processing/pre-aggregation needed for the visualization. Furthermore, no specific effort to optimize such runtimes was attempted, which suggests that some space for improvement is probably still available.

In terms of performance, our pipeline delivers promising results. The output clusters show an evident ability to capture both structural and semantic similarity between messages, as discussed in Sect. 3.1. This result is achieved despite applying minimal hard-coded feature engineering during pre-processing and exploiting simple models for vectorization and clustering. Interestingly, incorporating additional auxiliary information related to the source and destination hostnames seems to help unravel higher-level interactions between the nature of the issues and where they occur. This, in turn, provides a finer detail when spotting problems that may aid the human operators to restore the proper functioning of the infrastructure faster.

The previous considerations are also corroborated by a quantitative assessment of the pipeline's potential impact when applied to daily workflows. This is done by comparing the outputs of our approach to the incidents reported in GGUS in a reasonable time window around the day of the analysis. In terms of the direct association between clusters and tickets, the performance varies from average to decent depending on how much nuisance one is willing to tolerate in the output. Regarding the inverse relationship, instead, the approach is highly accurate since it highlights 8 out of the 9 incidents observable on the day of the analysis.

Nonetheless, some adjustment and tuning would be helpful prior to full integration into production. First, the analyzed clusters show indications that additional tuning may be needed to guarantee a more suitable level of granularity. This task is highly application-specific and requires the direct involvement of operators and site experts. A second concern is related to the limited number of errors shown. Ideally, the perfect output for our use case would be one error pattern—or even a more human-readable description directly pointing to the source of the problem—per cluster, for a small number of clusters (e.g. ≤ 6). In practice, however, the magnitude of the problem still refers to the actual number of failures. Even reducing it to the minimum, this is still bounded by the number of combinations between unique strings/patterns and source/destination locations, which is clearly overwhelming to handle for human operators. Therefore, the desired output is hardly deliverable as there is a trade-off between the clusters' internal homogeneity (number of patterns) and their number. For this reason, we reach a compromise by setting a higher value of k and displaying just a fixed, customizable portion of each cluster (three patterns in the current implementation). However, limiting the visualized patterns potentially hinders serious faults of medium and small sizes. Moreover, the necessity to mask message parameters to get more informative and abstract descriptions prevents using their values for troubleshooting—e.g. when the failures are due to specific parameter values. To comply with the above requirements, a possible solution is the implementation of a flexible and efficient user interface that allows the operators to adjust the number of displayed patterns and enables interactive drill-down to investigate more closely the effect of parametric values. Nevertheless, guaranteeing a good balance constitutes an intrinsic challenge of our use case, and its resolution again requires a direct tuning by experts.

Furthermore, although it makes sense to cross-check clustering results with GGUS tickets for a quantitative evaluation, this comparison has drawbacks. On one side, GGUS incidents force to focus solely on reported failures, thus preventing the study of undetected issues and masking some omission policies due to external factors— e.g. the site is in downtime or blacklisted, or the fault is known to be transient and therefore not reported. On the other side, the procedure is sensitive to the choice of the time window. Indeed the issues may have no match because they are reported before the selected period or due to delays in their discovery and reporting. All in all, the assessment may be biased because of these factors, thus limiting the reaches of the conclusions drawn. A better solution would be directly measuring the impact on workloads, which requires the involvement of operators and/or site experts for better tuning and use in production. For this reason, the idea behind our work is to

showcase a possible approach and hopefully stimulate further developments.

Finally, a potential issue may arise if the the error messages suddenly change in content or structure. However, this is not likely to happen inadvertently and should therefore be a minor concern. Moreover, a simple re-training of the vectorization stage should suffice in such circumstance, so that the updated embedding correctly represents the new message structure and semantic.

Future Work

All of the previous adjustments demand additional in-depth studies, each requiring a lengthy manual review of the results due to the unsupervised approach. Also, most of the above solicit direct participation of system experts to guarantee the soundness of the results and proper tuning. Considering the several appointed investigations and the conspicuous number of alternative combinations, it becomes clear how the requested effort is not sustainable and does not scale to the comparison of adversarial approaches. A possible solution we envision for future developments is represented by the collection of a **reference dataset** in which to store labels for error categories, root causes, incident priority and solving actions. In this way, the evaluation of new experiments would become immediate and systematic (e.g. [27]). Also, this would make the investigation of novel techniques sustainable, enlarging the plethora of applicable approaches to supervised methods and enabling a coherent comparison of alternative algorithms. Perhaps more importantly, the derived measure of performance would be linked to the actual goal of the analysis, thus allowing a direct optimization of the models for the specific task of interest.

Remaining in the unsupervised learning domain, several alternative approaches can be explored both regarding the vectorization stage [9, 15, 34] and for clustering [2, 19, 29, 33, 41]. Another interesting research line would be to explore end-to-end solutions that address both vectorization and clustering stages together. For example, one could try to directly optimize the learned embedding for the following clustering stage [40]. Alternatively, one can tap into topic modeling literature to jointly compute the vector representation of the messages and the derived topics (error categories in our use case) [8, 11, 23, 32].

Finally, a crucial contribution may come from the standardization of the error messages at source so to make them more consistent and explanatory. This would be helpful in two ways. On one side, more structured error templates would be easier to parse during the abstraction mechanism, thus improving the visualization of the results. On the other side, it would facilitate the analysis of the messages, perhaps allowing the adoption of simple heuristics for parsing them and questioning the need of an intelligent approach in the first place.

Acknowledgements This work was done as part of the distributed computing research and development programme within the ATLAS Collaboration, which we thank for their support.

Funding Open access funding provided by Alma Mater Studiorum - Università di Bologna within the CRUI-CARE Agreement.

Data availability statement The data that support the findings of this study are available upon reasonable request from the respective LHC Experiments.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Albalawi R, Yeap TH, Benyoucef M (2020) Using topic modeling methods for short-text data: a comparative analysis. Front Artif Intell. https://doi.org/10.3389/frai.2020.00042
2. Ankerst M, Breunig MM, Kriegel HP, Sander J (1999) Optics: ordering points to identify the clustering structure. SIGMOD Rec. 28(2):49–60. https://doi.org/10.1145/304181.304187
3. Antoni T, Bühler W, Dres H, Grein G, Roth M (2008) Global grid user support—building a worldwide distributed user support infrastructure. J Phys: Conf Ser 119(5):052002. https://doi.org/10.1088/1742-6596/119/5/052002
4. Arthur D, Vassilvitskii S (2007) K-means++: the advantages of careful seeding. In: In Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms
5. ATLAS Collaboration: The atlas experiment at the cern large hadron collider. Journal of instrumentation 3:S08003 (2008)
6. Barisits M, Beermann T, Berghaus F, Bockelman B, Bogado J, Cameron D, Christidis D, Ciangottini D, Dimitrov G, Elsing M et al (2019) Rucio: scientific data management. Comput Softw Big Sci 3(1):1–19
7. Bird I (2011) Computing for the large hadron collider. Ann Rev Nuclear Particle Sci 61:99–118. https://doi.org/10.1146/annurev-nucl-102010-130059
8. Blei DM, Ng AY, Jordan MI (2003) Latent Dirichlet allocation. J Mach Learn Res 3:993–1022
9. Brown TB, Mann B, Ryder N, Subbiah M, Kaplan J, Dhariwal P, Neelakantan A, Shyam P, Sastry G, Askell A et al (2020)

Language models are few-shot learners. arXiv preprint arXiv:2005.14165
10. Calafiura P, Catmore J, Costanzo D, Di Girolamo A (2020) Atlas hl-lhc computing conceptual design report. Tech. rep., CERN, Geneva. https://cds.cern.ch/record/2729668
11. Chen Y, Zhang H, Liu R, Ye Z, Lin J (2019) Experimental explorations on short text topic mining between lda and nmf based schemes. Knowl-Based Syst 163:1–13
12. Clissa L (2022) Survey of big data sizes in 2021
13. Decker L, Leite D, Giommi L, Bonacorsi D (2020) Real-time anomaly detection in data centers for log-based predictive maintenance using an evolving fuzzy-rule-based approach. In: 2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), pp. 1–8. https://doi.org/10.1109/FUZZ48607.2020.9177762
14. Decker L, Leite D, Viola F, Bonacorsi D (2020) Comparison of evolving granular classifiers applied to anomaly detection for predictive maintenance in computing centers. In: 2020 IEEE Conference on Evolving and Adaptive Intelligent Systems (EAIS), pp. 1–8. https://doi.org/10.1109/EAIS48028.2020.9122779
15. Devlin J, Chang MW, Lee K, Toutanova K (2018) Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805
16. Di Girolamo A, Legger F, Paparrigopoulos P, Schovancová J, Beermann T, Boehler M, Bonacorsi D, Clissa L, Decker-de-Sousa L, Diotalevi T, Giommi L, Grigorieva M, Giordano D, Hohn D, Javurek T, Jezequel S, Kuznetsov V, Lassnig M, Mageirakos V, Olocco M, Padolski S, Paltenghi M, Rinaldi L, Sharma M, Tisbeni SR, Tuckus N (2022) Preparing distributed computing operations for the hl-lhc era with operational intelligence. Front Big Data 4:115. https://doi.org/10.3389/fdata.2021.753409
17. Di Girolamo (2020) Alessandro, Legger, Federica, Paparrigopoulos, Panos, Klimentov, Alexei, Schovancová, Jaroslava, Kuznetsov, Valentin, Lassnig, Mario, Clissa, Luca, Rinaldi, Lorenzo, Sharma, Mayank, Bakhshiansohi, Hamed, Zvada, Marian, Bonacorsi, Daniele, Rossi Tisbeni, Simone, Giommi, Luca, Decker de Sousa, Leticia, Diotalevi, Tommaso, Grigorieva, Maria, Padolski, Sergey: Operational intelligence for distributed computing systems for exascale science. EPJ Web Conf. 245:03017. https://doi.org/10.1051/epjconf/202024503017
18. Diotalevi T, Bonacorsi D, Falabella A, Giommi L, Martelli B, Michelotto D, Morganti L, Rossi Tisbeni S (2019) Collection and harmonization of system logs and prototypal Analytics services with the Elastic (ELK) suite at the INFN-CNAF computing centre. In: Proceedings of International Symposium on Grids & Clouds 2019—PoS(ISGC2019), vol. 351, p. 027. https://doi.org/10.22323/1.351.0027
19. Ester M, Kriegel HP, Sander J, Xu X et al (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. KDD 96:226–231
20. Giommi L, Bonacorsi D, Diotalevi T, Rinaldi L, Morganti L, Falabella A, Ronchieri E, Ceccanti A, Martelli B, Tisbeni S (2019) Towards predictive maintenance with machine learning at the INFN-CNAF computing centre. In: Proceedings of International Symposium on Grids & Clouds 2019—PoS(ISGC2019), 351:003. https://doi.org/10.22323/1.351.0003
21. Giordano D (2021) Paltenghi, Matteo, Metaj, Stiven, Dvorak, Antonin: Anomaly detection in the cern cloud infrastructure. EPJ Web Conf. 251:02011. https://doi.org/10.1051/epjconf/202125102011
22. Grigorieva M, Grin D (2021) Clustering error messages produced by distributed computing infrastructure during the processing of high energy physics data. Int J Mod Phys A 36(10):2150070–130. https://doi.org/10.1142/S0217751X21500706
23. Grootendorst M (2022) Bertopic: Neural topic modeling with a class-based tf-idf procedure. https://doi.org/10.48550/ARXIV.2203.05794
24. Guyon I, Von Luxburg U, Williamson RC (2009) Clustering: science or art. In: NIPS 2009 workshop on clustering theory, pp. 1–11. Citeseer
25. Karavakis E, Manzi A, Rios MA, Keeble O, Cabot CG, Simon M, Patrascoiu M, Angelogiannopoulos A (2020) Fts improvements for lhc run-3 and beyond. In: EPJ Web of Conferences, vol. 245, p. 04016. EDP Sciences
26. Leite D, Decker L, Santana M, Souza P (2020) Egfc: Evolving gaussian fuzzy classifier from never-ending semi-supervised data streams - with application to power quality disturbance detection and classification. In: 2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), pp. 1–9. https://doi.org/10.1109/FUZZ48607.2020.9177847
27. Lin Q, Zhang H, Lou JG, Zhang Y, Chen X (2016) Log clustering based problem identification for online service systems. In: 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), pp. 102–111. IEEE
28. Lloyd S (1982) Least squares quantization in PCM. IEEE Trans Inf Theory 28(2):129–137
29. McInnes L, Healy J, Astels S (2017) hdbscan: hierarchical density based clustering. J Open Sourc Softw 2(11):205
30. Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781
31. Minarini F, Decker L (2020) Time-series anomaly detection applied to log-based diagnostic system using unsupervised machine learning approach. In: Conference of Open Innovations Association, FRUCT, 27, pp. 343–348. FRUCT Oy
32. Neogi PPG, Das AK, Goswami S, Mustafi J (2020) Topic modeling for text classification. Emerging technology in modelling and graphics. Springer, Berlin, pp 395–407
33. Ng AY, Jordan MI, Weiss Y (2002) On spectral clustering: analysis and an algorithm. In: Advances in neural information processing systems, pp. 849–856
34. Peters ME, Neumann M, Iyyer M, Gardner M, Clark C, Lee K, Zettlemoyer L (2018) Deep contextualized word representations. arXiv preprint arXiv:1802.05365
35. Rousseeuw PJ (1987) Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. J Comput Appl Math 20:53–65 https://doi.org/10.1016/0377-0427(87)90125-7. www.sciencedirect.com/science/article/pii/0377042787901257
36. Schovancová J (2019) Atlas computing operations. Tech. rep., ATLAS Collaboration. https://indico.cern.ch/event/809227/contributions/3370897/attachments/1820938/2978308/20190401-ATLAS_Computing_Operation_partial_view.pdf
37. Tisbeni SR (2019) Big data analytics towards predictive maintenance at the infn-cnaf computing centre. PhD thesis, University of Bologna. http://amslaurea.unibo.it/18430/
38. Von Luxburg U, Williamson RC, Guyon I (2012) Clustering: science or art? In: Proceedings of ICML workshop on unsupervised and transfer learning. JMLR Workshop and Conference Proceedings, pp. 65–79
39. Wold S, Esbensen K, Geladi P (1987) Principal component analysis. Chemom Intell Lab Syst 2(1–3):37–52
40. Yang B, Fu X, Sidiropoulos ND, Hong M (2017) Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In: International conference on machine learning, PMLR, pp. 3861–3870
41. Zhang T, Ramakrishnan R, Livny M (1996) Birch: an efficient data clustering method for very large databases. SIGMOD Rec. 25(2):103–114. https://doi.org/10.1145/235968.233324