Transaction aware tape-infrastructure monitoring

Fotios Nikolaidis, Daniele Francesco Kruse
E-mail: fotis.nikolaidis@cern.ch, daniele.francesco.kruse@cern.ch

Abstract. Administrating a large scale, multi protocol, hierarchical tape infrastructure like the CERN Advanced STORage manager (CASTOR)[2], which stores now 100 PB (with an increasing step of 25 PB per year), requires an adequate monitoring system for quick spotting of malfunctions, easier debugging and on demand report generation. The main challenges for such system are: to cope with CASTOR’s log format diversity and its information scattered among several log files, the need for long term information archival, the strict reliability requirements and the group based GUI visualization. For this purpose, we have designed, developed and deployed a centralized system consisting of four independent layers: the Log Transfer layer for collecting log lines from all tape servers to a single aggregation server, the Data Mining layer for combining log data into transaction context, the Storage layer for archiving the resulting transactions and finally the Web UI layer for accessing the information. Having flexibility, extensibility and maintainability in mind, each layer is designed to work as a message broker for the next layer, providing a clean and generic interface while ensuring consistency, redundancy and ultimately fault tolerance. This system unifies information previously dispersed over several monitoring tools into a single user interface, using Splunk, which also allows us to provide information visualization based on access control lists (ACL). Since its deployment, it has been successfully used by CASTOR tape operators for quick overview of transactions, performance evaluation, malfunction detection and from managers for report generation.

1. Introduction
CASTOR (CERN Advanced STORage manager) is a hierarchical storage system having already stored more than 100 PB and is continuously increasing (both in terms of hosted data and infrastructure size). This capacity is provided by a few thousands of disks and tens of thousands of tapes. Unsurprisingly, there are many reasons which can cause performance degradation in such a huge system. To list a few of those reasons, network failures, disk system congestion and tape library malfunctions are usual suspects.

Due to the randomness and complexity of these problems, it is impossible to foresee all of them and take proactive precautions. To overcome that we need a monitoring mechanism which can help us to go through reactive approach of solving the problems.

That mechanism must be flexible, user-friendly and able to combine information originated from multiple sources. In our case the sources are logs generated from tapeservers and the output of CASTOR standard tools.

For the tape infrastructure monitoring we have implemented a redundant, fault tolerant system consisting of multiple independent layers. The basic components are scripts for creating tape session statistics, a database responsible for storing the results and Splunk[1] for its visualisation capabilities. In addition, Splunk gives us an option of presenting different levels of
information depending on the user privileges user. Given that, it is possible to provide a more
detailed view to the operator, while providing a more user-friendly interface to the managers.

2. Monitoring
For proper monitoring we need two mechanisms. The first will give us tape transaction statistics
(Tapelog) and the other one will keep information for the tapes supplies (Warehouse) :

2.1. CASTOR TAPELOG
2.1.1. Collecting information During data movement either is disk to tape (migration) or from
tape to disk (recall) multiple logs are generated, each one describing the current status of the
running tape session (local knowledge). When the log entry generated, the machine handling the
tape session (Tapeserver) uses Rsyslog[4] utility to send a copy to a central server (Logserver)
and keeps a local copy.

2.1.2. Core Since each logline contains only local knowledge we have built a Perl daemon
tapelog-parser which is responsible for merging logs of the same origin (session) and return a
single log line describing the whole tape session.

The idea behind it is that loglines describing the same tape session have the same tapeserver
and jobid. These two fields which are subset of indexing information that every log carries are
used to build an index key in the hash table holding information about all the processes.

Given the possibility that the order or format of the log entry fields may change over time,
the parser is built flexible enough to allow to define fields of interest, aliases of these fields and
field separators.

The logs originated from the tapeserver include a code describing the current operation of
the tape session.

For every log of the same source that code is appended to a chain (a list under the same hash
key). The core of the parser is a Finite State Machine (FSM) which decides is a session is valid
(following predefined steps) or invalid (overlapping sessions, labelling etc.)

• valid and finished → Tape Transaction is marked as successful.
• valid and unfinished → Remains OPEN.
• valid and failed → Tape Transaction is marked as failed.
• invalid → Tape Transaction is dropped and all the information is dumped to a debug file.

If a tape session is OPEN and another session starts with the same hash key, instead of
overwriting we keep all the information and we mark the session as “conflict”. Then the tape
operator is responsible to find what is wrong.

When a transaction is marked as successful we print in key-value format all the information.

2.1.3. Extended information Even if we combine the logs to construct a tape session, the
resulting knowledge is only limited to lower layers of the infrastructure logic. More information
can be fetched from upper layers. For example, a tapeserver has knowledge regarding the
mounted tape and only that. But every tape is also associated with a tapepool. When we
construct a tape transaction, the corresponding tapepool can be fetched from standard CASTOR
utilites and be appended.

At the end of the parsing, we have extracted all the information we need for describing the
environment of a transaction. Some descriptive fields are

• Datavolume : carries the number of transferred data.
• Files : carries the number of transferred files.
(Un)mounttime / positiontime: Indicate the time needed to prepare a tape for an action.

Transfer-time: Is the actual time needed to transfer the given datavolume (without overhead).

Requeststate: Indicates if the transaction was successful / failed.

Tapepool: Indicates the logical group to which the tape is assigned.

Every tape session statistics is appended to a file and after a while a cron job is triggered and the statistics are pushed to the database. If the operation is successful the file is truncated. If it fails, the database is rolled back and file remains unchanged until the next hit of the cron job and a mail is sent to the operators.

2.2. DATA WAREHOUSE

2.2.1. Collecting information
Although keeping track of transactions can help us to spot malfunctions, we also need information for the tape supplies for capacity planning. This way we can estimate the occupied / freespaces, list of tapes per tapepools, file metadata, tape contents etc. The above information is provided by CASTOR standard tools like VMGR (Volume Manager), NS (Name Server) and REPACK. In contrast to logs the information provided by CASTOR tool comes in the table format which makes the information extraction rather easy.

2.2.2. Usage
These data are utilized by various scripts for forensics, daily reports, alerts and reacting on certain events. For example, if a tape has been mounted more than 100 times in less than a day that tape is disabled for security reasons. Another example is when a tape has errors. Using the previous knowledge we can easily backtrace the lost files by cross checking tape metadata with the Name Server.

Since fetching statistics for the whole infrastructure is time-consuming they are cached in a database to be accessible by applications that need them. Every 15 minutes this cache is updated with the latest records. For every record that is replaced we keep a copy of the old values. This way we form a history log for each tape which we can look through later.

3. Database

3.1. Normalization
Because of the nature of the logs, most of the fields have frequently repeated values. Therefore, the database schema is normalized by having each field represented as a table with the record in the logfile table keeping pointers to the values in the corresponding tables. This allows to reduce database size by eliminating storage of duplicate information.

3.2. Lookups
Lookups are also needed to connect logline records with other external information. An example of such information is Virtual Organization that corresponds to given tapepool. When a new record is added a trigger appends the necessary fields to the record.

4. Visualisation
Over the past of time, several monitoring tools, like LEMON[6] and APEX[5], have been introduced to CASTOR. Most of the times each of them served only a certain purpose, thus they were rigid with limited capabilities. As a result, graphs and statistics where spread over multiple interfaces and tools which did the event correlation almost impossible. To overcome that, we built a universal tape monitoring system on top of Splunk’s versatile interface engine, which also comes with group based information access.
4.1. Inputs
The first step is to import data from the database to Splunk. We can do it either by dumping database to a file and have Splunk index it, or by fetching information through scripts written in Splunk framework or through a dedicated module for Splunk. We use files for the synchronization, scripts for lookups and experimentally we are moving toward DBmodule.

4.2. Searches
In Splunk we can distinguish two different kinds of searches. One involves filtering out records and keeping only what we are interested in. The other is that we apply a function over fields and we get the result.

Usually in the first case we do not know a priori what we are looking for and searches must be performed at that time. These searches are characterized by the fact that they are performed fast, users have better control over the data and even the latest records are taken into account. We call them Real time searches.

For the second case we know what function we want to apply to which fields but usually it is time and resource consuming. To overcome that, we sacrifice the up-to-date results for better user experience and we schedule Saved searches to be calculated overnight.

For example, finding the data movement of a given tapepool for the last week takes just a few seconds while doing that over all time can take a few minutes. As an extra improvement on that we use accelerated reports. It relies on the same principles as checkpoints. The partial results of a search (as defined by function - field) are saved and at the end we just have to operate on the data from the latest ”checkpoint” until the end of the period that we are looking at.

4.3. AAA Protocol
A user session is following the AAA[8] protocol. After the user Authentication through SSO (Single Sign On) and LDAP (Lightweight Directory Access Protocol) -both provided as service by CERN- and Authorization (access to a certain level of information) based on the subscribed groups, the user has access to limited resources as opposed by resource Accounting mechanism.

4.4. Interface
AAA protocol makes explicit that each user has access to different information based on the subscribed groups. Since these information and their correlation are represented in different ways, different interfaces are needed for each user. Specifically we can separate the provided interfaces into 3 categories. The debug panel is used by operators. It is a collection of filters, functions, frequency tables and other tools that can help them. The overview panel is used by managers. Is a collection of commonly used graphs that someone can access without any knowledge of Splunk. The custom view is used when someone needs a custom graph. For these cases we have built a generic interface providing all the fields that the user can access and a few functions so that he can build the graph on his own.

Unfortunately, the aforementioned interfaces are designed to visualize parallel sessions and their duration. For that reason we had to built a Splunk module for visualising Gantt Charts.

5. Failure Analysis
As was mentioned before, our architecture consists of multiple independent layers. The layers follow the writeback principle that if lower layers cannot push data to upper layers, then they cache them and retry later.

- If Rsyslog crashes, logs are locally saved in tapeservers and can be resend later.
- If the Parser crashes, logs are locally saved in logserver and they can be parsed later.
• If the Loader crashes, extracted information is locally stored and can be pushed later to the database.
• If the Database crashes, information is stored in Splunk indexes.
• If Splunk crashes, information is stored in the database and can be re-indexed later.

6. Conclusions
Our system has been deployed and running in production for almost a year and since then it has successfully served tape operators to spot malfunction and other users to have the requested plots without any interaction with operators.

References
[1] Splunk official Website http://www.splunk.com/
[2] CASTOR official Website http://castor.web.cern.ch/
[3] Shibboleth official Website http://www.shibboleth.com/
[4] Rsyslog official Website http://www.rsyslog.com/
[5] APEX official Website http://apex.oracle.com/i/index.html
[6] Lemon official Website http://lemon.web.cern.ch/lemon/index.shtml/
[7] Oracle Documentation (Overview of Extraction Transformation and Loading).
[8] Cisco Documentation ( Triple-A protocol).
[9] F. Nikolaidis, L. Brarda, J-C Garnier, N. Neufeld. 2010 A universal logging system for LHCb Online. J. Phys.: Conf. Ser. 331 022037
[10] J-C Garnier, L. Brarda, N. Neufeld, F. Nikolaidis. 2011 LHCb Online Log Analysis and Maintenance System. Conf. Proc. C111010 (2011) pp.THCHAUST05