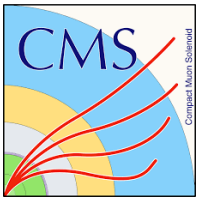


Enhancement of the CMS Tabular Data Monitoring Pipeline

CERN Summer Student: Zeina El Kojok

Summer Student 2024 Report submitted in partial fulfillment
of the requirements for the CERN Summer Student Program
on August 5, 2024

Team: CMS Monitoring

Project Advisors
Brij Kishor Jashal
Federica Legger
Nikodemas Tuckus

CERN
Geneva, Switzerland

Abstract

Monitoring of CMS distributed storage is paramount to ensure efficient use of disk and tape resources allocated to the experiment. The CMS Monitoring team currently uses MongoDB to store data for the tabular data monitoring and DataTables to visualize the data. However, this pipeline can be improved and streamlined to enhance the performance. This project aims to implement OpenSearch for storage and Grafana for access to overcome the performance issues and to unify the tabular data monitoring pipeline with other dataset monitoring pipelines.

Keywords: CMS, Tabular Data Monitoring, MongoDB, OpenSearch, Grafana, DataTables, PySpark

Acknowledgment

I would like to express my gratitude to my advisors, Brij Kishor Jashal, Federica Legger, and Nikodemas Tuckus, for guiding me throughout the different phases of the project. Their insights, support, and expertise were invaluable in the completion of this work. I also appreciate the resources and facilities provided by CERN, which made this work possible.

Contents

Chapter 1

Introduction

Efficient data storage and processing has always been a main concern for the LHC experiment at CERN especially following the LHC's current run, also known as Run 3. By the end of this run, the available dataset for the physics analysis is expected to triple. The Worldwide LHC Computing Grid (WLCG) is used by CMS to manage the workload and data. This grid consists of more than a hundred computing centers distributed all around the world. [1] [9]

The CMS Offline and Computing team (O&C) at CERN are responsible for ensuring appropriate levels of computing resources available for the CMS physics analysis and computing program. It ensures that the resources are being used efficiently with the software applications while also minimizing operational effort, and completing computing requests in short, predictable amounts of time. The O&C provides a computing model in which we can utilize processing (including accelerators and heterogeneous architectures, HPCs), disk and tape storage, and network in a flexible manner.[4]

Within the CMS O&C, the CMS Monitoring team ensures efficient use of disk and tape resources allocated to the experiment by monitoring the different datasets. The CMS experiment relies on Rucio and DBS (Dataset Bookkeeping service) to handle data. [1]

The distributed storage managed by Rucio is monitored through the tabular dataset monitoring. The tabular dataset monitoring consists of tables summarizing for each dataset the content, size, creation and last access dates, and location on the CMS Rucio Storage Elements (RSEs). The CMS Monitoring team currently uses MongoDB to store data for the tabular data monitoring and DataTables to visualize the data. However, this pipeline can be improved and streamlined to enhance the performance. [1]

This project aims to implement OpenSearch for storage and Grafana for access to overcome the performance issues and to unify the tabular data monitoring pipeline with other dataset monitoring pipelines.

Chapter 2

Current Tabular Dataset Monitoring

2.1 Shared Pipeline Steps

The CMS Monitoring team combines Rucio and DBS data sources to obtain information about the stored data in order to ensure efficient use of disk and tape resources allocated to the experiment. CMS uses Rucio as its distributed data management of choice for the data. Rucio stores information about the location of the data and includes details about the dataset names, list of files belonging to each dataset, size, creation date, last access date, policy on the number of copies to keep on different types of storage (tape or disk). Meanwhile, the DBS stores details about the dataset's metadata, such as data tier, production campaign, and acquisition era. Custom ETL (Extract, Transform, Load) pipelines are used for various needs since Rucio and DBS databases can not be queried directly for performance reasons.[1]

Most of these pipelines share the steps below:

- Relevant database tables are extracted on a daily basis, metrics of interest are aggregated, and displayed in a variety of views. The first part of the data flow, namely the extraction and aggregation from the Rucio and DBS databases, is common to the three pipelines.
- Sqoop jobs are used to create the daily DB snapshots that are stored in HDFS. The Sqoop jobs are optimized and executed in parallel, query about ten DB tables, and typically last less than thirty minutes.
- Snapshots in HDFS are processed by Spark jobs that aggregate metrics of interest for each specific use case. Each Spark job consists of more than twenty join operations between Spark datasets and tens of aggregations to reach the final data schema.

2.2 PySpark Scripts and Cron Jobs used for Tabular Dataset Monitoring

Two main PySpark scripts are used to aggregate the data for the tabular data monitoring.

The first script datasets.py creates datasets summary results by aggregating Rucio and DBS tables and saves result to HDFS directory as a source to MongoDB of go web service. This scripts aggregates 12 dataframes by performing different join operations and saves the final dataframe in the main subdirectory on HDFS.[7]
The second script detailed_datasets.py creates detailed datasets(in each RSEs) results by also aggregating Rucio and DBS tables and saves the result to HDFS directory as a source to MongoDB of go web service This script performs 15 aggregation, most of which are join operations, and saves the final results to the detailed and the in_tape_and_disk subdirectories.[8]

Both of the PySpark scripts are actually called by the cron4rucio_spark2hdfs.sh cron job that is scheduled to run once daily. This job runs the Spark jobs to get Rucio and DBS datasets and writes to the HDFS directory.[6]
Directly after the first cron job, the cron4rucio_hdfs2mongo.sh cron job runs to get the HDFS results to a local directory and to send these datasets to MongoDB.[5]

2.3 DataBase and Visualization Used for the Tabular Dataset Monitoring

Following these aggregations, the data for the tabular dataset monitoring are injected in a standalone MongoDB instance that serves as a storage to a web service backed by Go and JQuery DataTables stack.[3]

This web page visualizes three tables:

- Main datasets table with over 1.5 million entries
- Datasets in Both Tape and Disk Table with over 550 thousand entries
- Detailed Datasets Table with over 6 million entries

The complete pipeline for the aggegation, storage and visualization of the tabular data monitoring is showin in Figure 2.1.

The DataTables web page shows the actual tabular dataset monitoring data as is and provides search capabilities for all the tables with different filter options. The tables are shown in Figures 2.2-2.5.

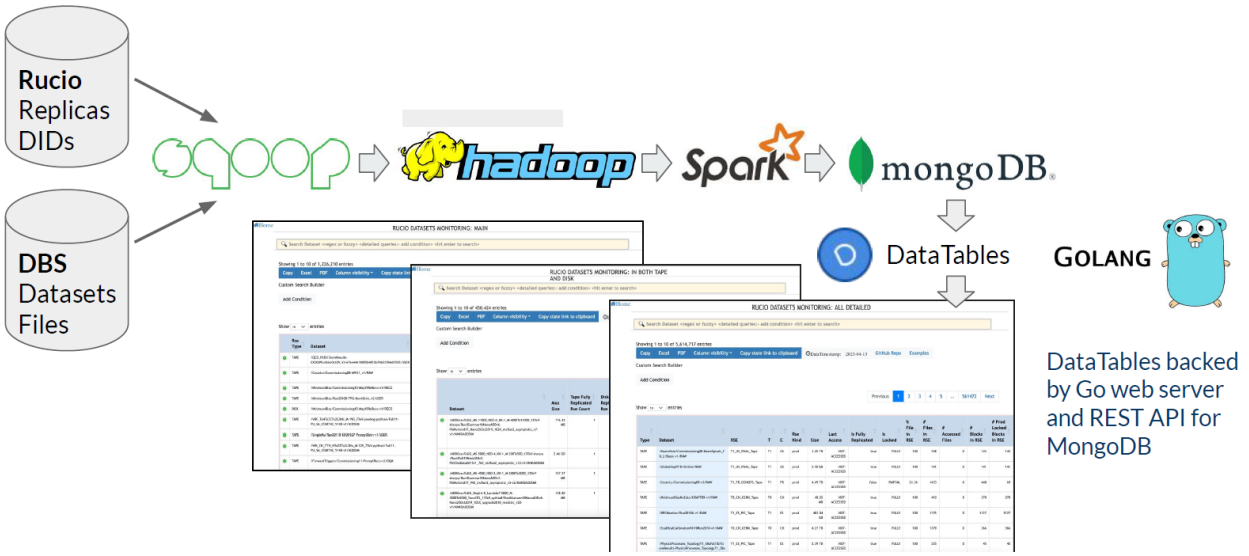


Figure 2.1: Current Tabular Dataset Monitoring Pipeline

	Rse Type	Dataset	Last Access	Max	Min	Avg	Sum	RealSize	FileCnt	RSEs
	TAPE	/ForwardTriggers/Commissioning11-PromptReco-v3/DQM	NOT-ACCESSED	798.93 MB	798.93 MB	798.93 MB	798.93 MB	798.93 MB	173	T0_CH_CERN_Tape
	TAPE	/QCD_Pt30/StoreResults-EXODiPhoSkimOct09_V3-676e44618905b4813b74b51ff84e07bf1/USER	NOT-ACCESSED	1.85 GB	1.85 GB	1.85 GB	1.85 GB	1.85 GB	1	T1_IT_CNAF_Tape
	TAPE	/MinimumBias/Commissioning10-May19ReReco-v1/RECO	NOT-ACCESSED	5.53 TB	5.53 TB	5.53 TB	5.53 TB	5.53 TB	1452	T1_DE_KIT_Tape
	DISK	/MinimumBias/Commissioning10-May19ReReco-v1/RECO	NOT-ACCESSED	5.53 TB	5.53 TB	5.53 TB	5.53 TB	5.53 TB	1452	T3_CH_CERN_OpenData
	TAPE	/SingleMu/Run2011B-EXOHSCP-PromptSkim-v1/USER	NOT-ACCESSED	597.12 GB	597.12 GB	597.12 GB	597.12 GB	597.12 GB	1035	T1_ES_PIC_Tape
	TAPE	/SingleMu/Run2011A-16Sep2011-ReHLT-v2/USER	NOT-ACCESSED	371.48 GB	371.48 GB	371.48 GB	371.48 GB	371.48 GB	369	T1_US_FNAL_Tape
	TAPE	/WprimeToTauNu_M-2700_TuneZ2_7TeV-pythia6-tauola/Fall11-PU_S6_START44_V5-v1/AODSIM	NOT-ACCESSED	3.47 GB	3.47 GB	3.47 GB	3.47 GB	3.47 GB	2	T0_CH_CERN_Tape
	TAPE	/Cosmics/CommissioningO8-WW31_v1/RAW	NOT-ACCESSED	425.92 GB	425.92 GB	425.92 GB	851.84 GB	425.92 GB	110	T0_CH_CERN_Tape, T1_ES_PIC_Tape
	TAPE	/MinimumBias/Run2010B-TPG-Nov6Skim_v2/USER	NOT-ACCESSED	1.58 TB	1.58 TB	1.58 TB	1.58 TB	1.58 TB	887	T1_US_FNAL_Tape
	TAPE	/HcalHPDNoise/PhRun2011-v1/RAW	NOT-ACCESSED	1.78 MB	1.78 MB	1.78 MB	1.78 MB	1.78 MB	10	T0_CH_CERN_Tape

Figure 2.2: Main Datasets Table

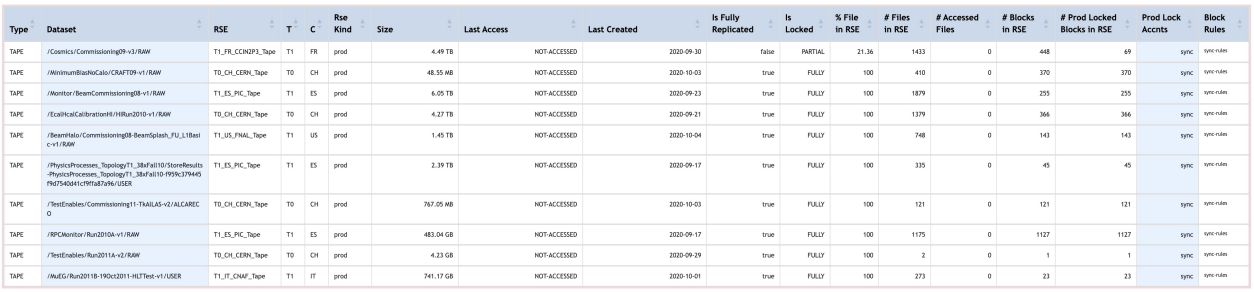
	Dataset	Max Size	Tape Fully Replicated Rse Count	Disk Fully Replicated Rse Count	Tape Fully Locked Rse Count	Disk Fully Locked Rse Count	Tape Rse Count	Disk Rse Count	Tape Rse Set	Disk Rse Set
	/ADDGravToZG_NegInt-0_LambdaT-6000_M-5007To1000_TuneCP5_13p6TeV_pythia8/Run3Summer22EEMiniAODv4-130X_mcRun3_2022_realistic_postEE_v6-v2/MINIAODSIM	19.64 GB	1	2	1	2	1	2	T1_US_FNAL_Tape	T1_FR_CCIN2P3_Disk, T2_US_Caltech
	/ADDGravToGG_MS-11000_NED-4_KK-1_M-4000To11000_13TeV-sherpa/RunIISummer16NanoAODv6-PUMoriond17_Nano25Oct2019_102X_mcRun2_asymptotic_v7-v1/NANOADDSIM	176.33 MB	1	6	1	6	1	6	T1_US_FNAL_Tape	T1_IT_CNAF_Disk, T1_UK_RAL_Disk, T2_CH_CERN, T2_EE_Estonia, T2_US_Purdue, T2_US_UCSD
	/ADDGravToGG_MS-4500_NED-2_KK-1_M-2000To3000_13TeV-sherpa/RunIISummer16NanoAODv3-PUMoriond17_94X_mcRun2_asymptotic_v3-v2/NANOADDSIM	157.17 MB	1	2	1	2	1	2	T0_CH_CERN_Tape	T2_FR_GRIF, T2_US_Nebraska
	/ADDGravToZG_NegInt-0_LambdaT-10000_M-2000To4000_TuneCP5_13p6TeV_pythia8/Run3Summer22DRPremix-124X_mcRun3_2022_realistic_v12-v2/AODSIM	37.41 GB	1	1	1	0	1	17	T1_US_FNAL_Tape	T1_DE_KIT_Disk, T1_FR_CCIN2P3_Disk, T1_IT_CNAF_Disk, T1_US_FNAL_Disk, T2_DE_DESY, T2_DE_RWTH, T2_ES_CIEMAT, T2_FI_HIP, T2_FR_GRIF, T2_IT_Bari, T2_IT_Rome, T2_PL_Swierk, T2_RU_JINR, T2_UK_London_IC, T2_UK_SGrid_RALPP, T2_US_Caltech, T2_US_Nebraska
	/ADDGravToGG_MS-5000_NED-2_KK-1_M-1000To2000_13TeV-sherpa/RunIISummer16NanoAODv3-PUMoriond17_94X_mcRun2_asymptotic_v3-v2/NANOADDSIM	144.33 MB	1	2	1	2	1	2	T0_CH_CERN_Tape	T1_UK_RAL_Disk, T1_US_FNAL_Disk
	/ADDGravToGG_MS-5000_NED-4_KK-1_M-200To500_13TeV-sherpa/RunIIFall15MiniAODv2-PU25nsData2015v1_76X_mcRun2_asymptotic_v12-v1/MINIAODSIM	2.66 GB	1	1	1	0	1	1	T0_CH_CERN_Tape	T3_CH_CERN_OpenData
	/ADDGravToGG_NegInt-0_LambdaT-5000_M-3000To5000_TuneCP2_13TeV-pythia8/RunIIAutumn18NanoAODv6-Nano25Oct2019_102X_upgrade2018_realistic_v20-v1/NANOADDSIM	138.30 MB	1	6	1	6	1	6	T1_US_FNAL_Tape	T1_FR_CCIN2P3_Disk, T1_US_FNAL_Disk, T2_BE_IIHE, T2_TW_NCHC, T2_UK_London_IC, T2_US_Wisconsin
	/ADDGravToGG_MS-7000_NED-4_KK-1_M-4000To7000_13TeV-sherpa/RunIISummer16NanoAODv6-PUMoriond17_Nano25Oct2019_102X_mcRun2_asymptotic_v7-v1/NANOADDSIM	185.69 MB	1	6	1	6	1	6	T1_US_FNAL_Tape	T1_DE_KIT_Disk, T1_IT_CNAF_Disk, T1_US_FNAL_Disk, T2_RU_INR, T2_UK_SGrid_RALPP, T2_US_Wisconsin
	/ADDGravToZG_NegInt-1_LambdaT-10000_M-4000To10000_TuneCP5_13p6TeV_pythia8/Run3Summer22EEMiniAODv4-130X_mcRun3_2022_realistic_postEE_v6-v2/MINIAODSIM	20.61 GB	1	1	1	1	1	28	T1_US_FNAL_Tape	T1_DE_KIT_Disk, T1_ES_PIC_Disk, T1_FR_CCIN2P3_Disk, T1_IT_CNAF_Disk, T1_RU_JINR_Disk, T1_UK_RAL_Disk, T1_US_FNAL_Disk, T2_AT_Vienna, T2_BE_IIHE, T2_BE_SPRACE, T2_CH_CSCS, T2_DE_DESY, T2_DE_RWTH, T2_EE_Estonia, T2_ES_CIEMAT, T2_FR_GRIF, T2_FR_IPHC, T2_IT_Bari, T2_IT_Legnaro, T2_IT_Rome, T2_PL_Cyfronet, T2_UK_London_IC, T2_UK_SGrid_RALPP, T2_US_Caltech, T2_US_Florida, T2_US_Nebraska, T2_US_UCSD, T2_US_Wisconsin
	/ADDGravToGG_NegInt-1_LambdaT-4000_M-3000To4000_TuneCP2_13TeV-pythia8/RunIIAutumn18NanoAODv7-Nano02Apr2020_102X_upgrade2018_realistic_v21-v1/NANOADDSIM	171.12 MB	1	6	1	6	1	6	T1_DE_KIT_Tape	T1_RU_JINR_Disk, T1_US_FNAL_Disk, T2_AT_Vienna, T2_BE_IIHE, T2_CH_CSCS, T2_US_Nebraska

Figure 2.3: Datasets in Both Tape&Disk Table

2.4 Issues with the Current Tabular Dataset Monitoring Pipeline

The current tabular dataset monitoring pipeline is facing several issues including:

- High maintainability efforts: Although the filtering and search capabilities of the current DataTables web service are sufficient, the main drawback of this pipeline the required main-



Type	Dataset	RSE	T	C	Rse Kind	Size	Last Access	Last Created	Is Fully Replicated	Is Locked	% File in RSE	# Files in RSE	# Accessed Files	# Blocks in RSE	# Prod Locked Blocks in RSE	Prod Lock Accnts	Block Rules
TAPE	/Cosmics/CommissioningO9-v3/RAW	T1_FR_CCIN2P3_Tape	T1	FR	prod	4.49 TB	NOT-ACCESSED	2020-09-30	false	PARTIAL	21.36	1433	0	448	69	sync	sync rules
TAPE	/MinimumBiasNoCalo/CRAFT09-v1/RAW	T0_CH_CERN_Tape	T0	CH	prod	48.55 MB	NOT-ACCESSED	2020-10-03	true	FULLY	100	410	0	370	370	sync	sync rules
TAPE	/Monitor/BeamCommissioning08-v1/RAW	T1_ES_PIC_Tape	T1	ES	prod	6.05 TB	NOT-ACCESSED	2020-09-23	true	FULLY	100	1879	0	255	255	sync	sync rules
TAPE	/EcalHcalCalibrationHI/HIRun2010-v1/RAW	T0_CH_CERN_Tape	T0	CH	prod	4.27 TB	NOT-ACCESSED	2020-09-21	true	FULLY	100	1379	0	366	366	sync	sync rules
TAPE	/BeamHalo/CommissioningO8-BeamSplash_FU_L1Basic-v1/RAW	T1_US_FNAL_Tape	T1	US	prod	1.45 TB	NOT-ACCESSED	2020-10-04	true	FULLY	100	748	0	143	143	sync	sync rules
TAPE	/PhysicsProcesses_Topology/T1_38xFall10/StoreResults-PhysicsProcesses_Topology/T1_38xFall10-f959c379445f9d754044fc9ffa87a9f6/USER	T1_ES_PIC_Tape	T1	ES	prod	2.39 TB	NOT-ACCESSED	2020-09-17	true	FULLY	100	335	0	45	45	sync	sync rules
TAPE	/TestEnables/CommissioningI1-TkAlLAS-v2/ALCARECO	T0_CH_CERN_Tape	T0	CH	prod	767.05 MB	NOT-ACCESSED	2020-10-03	true	FULLY	100	121	0	121	121	sync	sync rules
TAPE	/RPCMonitor/Run2010A-v1/RAW	T1_ES_PIC_Tape	T1	ES	prod	483.04 GB	NOT-ACCESSED	2020-09-17	true	FULLY	100	1175	0	1127	1127	sync	sync rules
TAPE	/TestEnables/Run2011A-v2/RAW	T0_CH_CERN_Tape	T0	CH	prod	4.23 GB	NOT-ACCESSED	2020-09-29	true	FULLY	100	2	0	1	1	sync	sync rules
TAPE	/MuEG/Run2011B-190ct2011-HLTTest-v1/USER	T1_IT_CNAF_Tape	T1	IT	prod	741.17 GB	NOT-ACCESSED	2020-10-01	true	FULLY	100	273	0	23	23	sync	sync rules

Figure 2.4: Detailed Datasets Table

tainability efforts of the stand-alone MongoDB and the DataTables especially since all other dataset monitoring pipelines do not use any similar MongoDB instance.

- Lack of any alarm system: Unless the DataTables are checked daily, errors related to data aggregation, sending to HDFS, and visualizing the data can go unnoticed since no current alarm system is implemented and no automated health check exists.

- Visualization of the values and not the relevant benchmarks: The search and filtering of the data are fast since the MongoDB instance has its own separate storage. However, the DataTables show the actual values of the fields and not the relevant metrics needed by the CMS Monitoring team.

Instead, it is desired to have a pipeline for the tabular dataset monitoring that resembles the othe dataset monitoring pipelines which use different storage technologies and have more user-friendly and visually appealing dashboards for visualizing the relavent metrics using charts.

Chapter 3

Environmental Setup

All the work was done on the test DM-MONIT Kubernetes cluster hosted on CERN cloud resources. The DM-MONIT hosts services dedicated to the data management monitoring.

A test master node and two test nodes are used for this cluster. The rucio-dataset-monitoring directory is added to the test cluster which contains all the required PySpark scripts and cron jobs. The required secrets and services are all deployed and the needed cron jobs are set up.
The test nodes are connected to a test web url for the DataTables.
The connection to the Hadoop HTCondor Analytix HDFS was established. SWAN is used to run the updated Spark jobs that push the data to hadoop and eventually OpenSearch.

Chapter 4

Methodology

4.1 Desired Tabular Data Monitoring Pipeline

As an attempt to unify the tabular dataset monitoring pipeline with the other dataset monitoring pipelines, the desired updated pipeline would use OpenSearch for storage instead of MongoDB and would use Grafana Dashboards for the visualization of the data instead of the DataTables web service backed up by GoLang.
Figure 4.1 shows the desired pipeline that should solve the shortcomings of the current pipeline and make the pipeline as similar as possible to the other dataset monitoring pipelines.

Figure 4.1: Desired Tabular Dataset Monitoring Pipeline

The ActiveMQ is not implemented on the test cluster since it can only be used for production purposes after the required OpenSearch production indices are created by the CERN IT department.

4.2 Changes to the PySpark Jobs

As an attempt to optimize the PySpark scripts, a thorough performance study was performed on both the datasets.py and detailed_datasets.py scripts.

Spark performs lazy evaluation, meaning it doesn't execute transformations immediately. Instead, it builds an execution plan called a Directed Acyclic Graph (DAG) which is only triggered when an action is called. Such actions include the like, collect, show and save commands.
Calling the functions for the dataframes can be either sequential or nested.

Sequential calling is when functions are called one after the other while nested calling is when the functions are called within each other.
Below is an example that highlights the key differences between both:

Sequential Dataframe Cells

```
# Main function
df1 = df1_fun(spark)
df2 = df2_fun(df1)
df3 = df3_fun(df2)
df3.show()

# Function definitions
def df1_fun(spark):
    # Create or read df1
    return df1

def df2_fun(df1):
    # Create df2 based on df1
    return df2

def df3_fun(df2):
    # Create df3 based on df2
    return df3
```

Nested Dataframe Cells

```
# Main function
df3 = df3_fun(spark)
df3.show()

# Function definitions
def df1_fun(spark):
    # Create or read df1
    return df1

def df2_fun(spark):
    df1 = df1_fun(spark)
    # Create df2 based on df1
    return df2

def df3_fun(spark):
    df2 = df2_fun(spark)
    # Create df3 based on df2
    return df3
```

The dataset.py script is a sequential dataframe script that utilizes 12 dataframes that go through

different join operation to reach the final dataframe. Whereas, the detailed_dataset.py script is a nested datafram script that used 15 dataframes to get the two final dataframes.
The main considerations when it comes to the PySpark performance in reference to the datasets.py and detailed_datasets.py scripts include:

1. Execution Plan and Optimization: Sequential dataframe calling has an explicitly constructed logical plan in the main function. Each DataFrame transformation is defined step-by-step. Spark's optimizer can see the entire chain of transformations at once and optimize the execution plan accordingly. As for the nested dataframe calling, the logical plan is constructed in a nested manner. Each function call constructs part of the logical plan, and Spark needs to understand these nested calls to optimize the entire job. Spark can still optimize this, but the visibility of the full transformation chain is less clear.

2. Readability and Maintenance: Sequential calling is more straightforward and readable. The transformation sequence is clear, making it easier to understand, debug, and maintain. However, sequential calling is less readable due to nested function calls. It might be harder to trace the sequence of transformations, which could complicate debugging and maintenance.

3. Data Storage and Reuse: Intermediate DataFrames can be explicitly cached or persisted in sequential calling, which can improve efficiency if those DataFrames are used multiple times in the job. Whereas in nested calls, intermediate DataFrames are created within function calls, so they are more transient. If df1 or df2 are used in multiple parts of the job, you might end up recomputing them multiple times, leading to inefficiency.

Therefore, the sequential datasets.py offers better control over performance and storage, making it easier to optimize and manage the Spark job compared to the detailed_datasets.py.

Accordingly, the detailed_dataset.py was updated and tested such that the complex script is more readable and it follows the same sequential structure of the datasets.py script.[2]

Chapter 5

Results

5.1 Database Change from MongoDB to OpenSearch

The first step to optimize the current pipeline is to change the dataset monitoring storage used. As an attempt to make this pipeline similar to other dataset monitoring pipelines, our go-to option for data storage is OpenSearch.
To do so, the PySpark jobs are edited such that the data is now sent to OpenSearch instead of MongoDB.[2] The first cronjob, cron4rucio_spark2hdfs.sh cron job, runs normally where both PySpark scripts are run. This cronjob saves the datasts in HDFS and sends the data of the tabular dataset monitoring directly to OpenSearch.
This eliminates the need for the cron4rucio_hdfs2mongo.sh cron job that was originally used to send the data to MongoDB.

The test OpenSearch is used to create three test indices: testzeina_main, testzeina_detailed and testzeina_t_and_d. These three indices store the data of the three final datasets that are the outputs of the datasets.py and detailed_datasets.py scripts.
The schema for each index is based on the json file entries in the HDFS and the corresponding index is defined accordingly. Figures 5.1-5.3 show the different schemas of each index and verifies that the data is actually being sent by the Spark jobs and received by OpenSearch successfully.

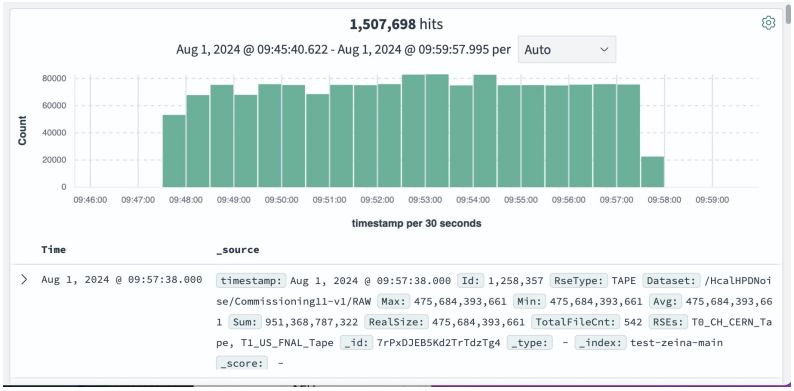
To avoid any resource limitation issue, the data is sent to OpenSearch in partitions such the resources do not get overwhelmed by the amount of data received.

5.2 Visualization Change from DataTables to Grafana

The second step is to visualize the data using Grafana. After the indices are actually up and running and are receiving the data, Grafana can be connected to the specific index. Each index is defined as a data source for the different Grafana Dashboards.

```python
def get_index_schema_main():
    return {
        "settings": {
            "index": {
                "number_of_shards": 1,
                "number_of_replicas": 1
            }
        },
        "mappings": {
            "properties": {
                "Id": {"type": "integer"},
                "RseType": {"type": "keyword"},
                "Dataset": {"type": "text"},
     #           "LastAccess": {"type": "long", "null_value": nan},
                "Max": {"type": "long"},
                "Min": {"type": "long"},
                "Avg": {"type": "long"},
                "Sum": {"type": "long"},
                "RealSize": {"type": "double"},
                "TotalFileCnt": {"type": "integer"},
                "RSEs": {"type": "keyword"}
            }
        }
    }
```

(a) Main Dataset Schema

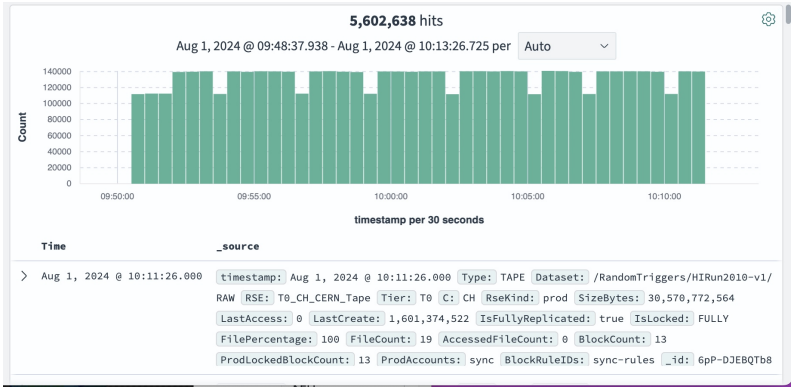


(b) Main Dataset OpenSearch Dashboard

Figure 5.1: Main Dataset OpenSearch Dashboard

```python
def get_index_schema_detailed():
    return {
        "settings": {
            "index": {
                "number_of_shards": 1,
                "number_of_replicas": 1
            }
        },
        "mappings": {
            "properties": {
                "Type": {"type": "keyword"},
                "Dataset": {"type": "text"},
                "RSE": {"type": "keyword"},
                "Tier": {"type": "keyword"},
                "C": {"type": "keyword"},
                "RseKind": {"type": "keyword"},
                "SizeBytes": {"type": "long"},
                "LastAccess": {"type": "long"},
                "LastCreate": {"type": "long"},
                "IsFullyReplicated": {"type": "boolean"},
                "IsLocked": {"type": "keyword"},
                "FilePercentage": {"type": "float"},
                "FileCount": {"type": "integer"},
                "AccessedFileCount": {"type": "integer"},
                "BlockCount": {"type": "integer"},
                "ProdLockedBlockCount": {"type": "integer"},
                "ProdAccounts": {"type": "keyword"},
                "BlockRuleIDs": {"type": "keyword"}
            }
        }
    }
```

(a) Detailed Dataset Schema

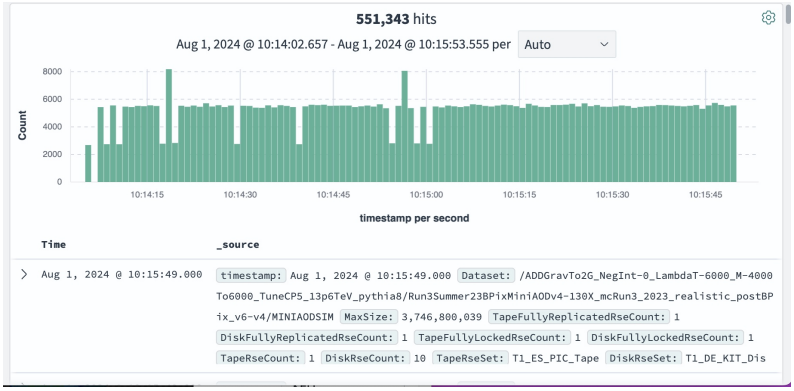


(b) Detailed Dataset OpenSearch Dashboard

Figure 5.2: Detailed Dataset

```python
def get_index_schema_t_and_d():
    return {
        "settings": {
            "index": {
                "number_of_shards": 1,
                "number_of_replicas": 1
            }
        },
        "mappings": {
            "properties": {
                "Dataset": {"type": "text"},
                "MaxSize": {"type": "long"},
                "TapeFullyReplicatedRseCount": {"type": "integer"},
                "DiskFullyReplicatedRseCount": {"type": "integer"},
                "TapeFullyLockedRseCount": {"type": "integer"},
                "DiskFullyLockedRseCount": {"type": "integer"},
                "TapeRseCount": {"type": "integer"},
                "DiskRseCount": {"type": "integer"},
                "TapeRseSet": {"type": "keyword"},
                "DiskRseSet": {"type": "keyword"}
            }
        }
    }
```

(a) In_tape_and_disk Dataset Schema



(b) In_tape_and_disk Dataset OpenSearch Dashboard

Figure 5.3: In_tape_and_disk Dataset

Different charts can be added by users which show some important metrics and benchmarks needed by the team. Filtering and grouping operations are also possible and allow the formation of complex queries.

Figure 5.4 shows a sample Grafana dashboard for the main dataset.

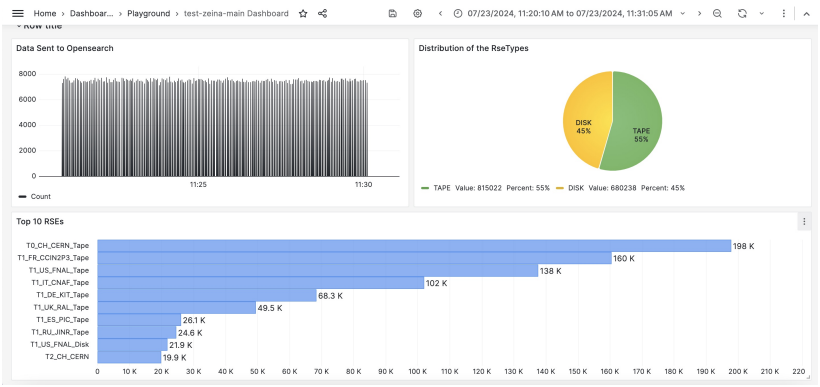


Figure 5.4: Sample Grafana Dashboard for the Main Dataset

5.3 Final Pipeline Advantages

Following the change of the tabular dataset monitoring storage from MongoDB to OpenSearch and the visualization change from DataTables to Grafana, all the issue with the initial pipeline have been addressed.

Some of the advantages of using OpenSearch and Grafana in the new pipeline include:

- facilitates health checks
- reduces maintenance efforts
- allows the implementation of an alert system
- user-friendly and visually appealing
- provides a unified dashboard platform for all monitoring pipelines
- allows the creation of charts with very specific aggregations to visualize metrics commonly used by the monitoring team

Chapter 6

Conclusion

To improve the performance of the tabular data monitoring pipeline, data storage has been moved from MongoDB to OpenSearch and the data visualization has been moved from DataTables to Grafana.
This change aligns with the other dataset monitoring pipelines as an attempt to reduce the maintainability efforts needed for the initial tabular dataset monitoring piepline and to visualize the data better.
The first part of the dataset related to the Rucio and DBS aggregation, daily snapshots and the PySpark join operations have been preserved. The PySpark jobs have been studied and optimized to ensure a better workflow of the pipeline.

Future work include the validation of the data in OpenSearch by making sure that the data is the same as that in MongoDB, fully reproduce the current functionalities of the DataTables in the new Grafana dashboards for the tabular dataset visualization, and perform performance checks for the Grafana queries to ensure that the dashboards load fast enough. Finally, the indices are now ready to be moved from the test to the production environment and the ActiveMQ step can be added just before sending the data to OpenSearch.

Appendix A

Project Poster

Project Poster presented on July 25th, 2024 at CERN as part of the Summer Student Poster Session 2024.

Bibliography

[1] Brij Kishor Jashal et al. "The CMS monitoring applications for LHC Run 3". In: *Proceedings of the Conference*. CHEAP 2023. Norfolk, Virginia, USA: 26th International Conference on Computing in High Energy 5 Nuclear Physics (CHEP2023), May 2023.

[2] Zeina El Kojok. *CMS Monitoring Project*. Accessed: August 4, 2024. URL: `https://github.com/zeinaelkojok/CMS_setup`.

[3] CMS O&C. *Data Management Monitoring*. Accessed: August 4, 2024. URL: `https://cms-dm-monitoring.cern.ch/`.

[4] Based on slides written by previous O&C Coordinators. "Offline and Computing". In: *Proceedings of the Conference*. 17th CMS Induction Course. Conference Presentation. CERN, Geneva, Switzerland, 2023.

[5] Ceyhun Uzunoglu. $cron4rucio_hdfs2mongo.sh$. GitHub repository script. 2022. URL: `https://github.com/dmwm/CMSMonitoring/blob/a4792c563e922c0dc42b59df56166f8bd69d5c54/rucio-dataset-monitoring/spark/cron4rucio_hdfs2mongo.sh`.

[6] Ceyhun Uzunoglu. $cron4rucio_spark2hdfs.sh$. GitHub repository script. 2022. URL: `https://github.com/dmwm/CMSMonitoring/blob/a4792c563e922c0dc42b59df56166f8bd69d5c54/rucio-dataset-monitoring/spark/cron4rucio_spark2hdfs.sh`.

[7] Ceyhun Uzunoglu. *datasets.py*. GitHub repository script. 2022. URL: `https://github.com/dmwm/CMSMonitoring/blob/a4792c563e922c0dc42b59df56166f8bd69d5c54/rucio-dataset-monitoring/spark/datasets.py`.

[8] Ceyhun Uzunoglu. *datasets.py*. GitHub repository script. 2022. URL: `https://github.com/dmwm/CMSMonitoring/blob/a4792c563e922c0dc42b59df56166f8bd69d5c54/rucio-dataset-monitoring/spark/detailed_datasets.py`.

[9] Ceyhun Uzunoglu et al. "Monitoring CMS experiment data and infrastructure for the next generation of LHC run". In: *Proceedings of the Conference*. ACAT 2022. Bari, Italy: ACAT 2022, Oct. 2022.