Analysis of transfers and requests in dCache using Apache Spark

Nataliia Derevesnikova
University of Tyumen, Russian Federation

September 5, 2018

Abstract

`dCache` is a storage system, which has an additional aim to host hundreds petabytes of data. This report represents the result of the analysis of the `dCache` billing data at DESY using `Apache Spark`, which can be used for the future upgrading of the mass storage system.

Supervisors: **Christian Voss**
Thomas Hartmann

Contents

1 Motivation

At the moment the log information from all access events (transfer, request, remove) in `dCache` is archived by day by department (ATLAS, CMS, etc.) in billing files, which are similar to text files, so proper reading of it usually takes a few hours for one day by one person, because of a several hundreds terabyte of data, where one file has a size from 1GB. But it's important to ensure disruption free operation of `dCache` by analysis the transfer records for errors.

2 Methods of analysis

In order to analyze the billing data, the contained information needs to be converted into more convenient view. So first, data is taken by using `Apache Spark` and its RDD (explanation of this framework is mentioned below) from files by lines and then put into labeled columns in a dataframe (pyspark.rdd.PipelinedRDD), from which then it can be selected using SQL commands. All computations are done using Python 3 and Jupyter Notebook (it's an open-source web application, to learn more: http://jupyter.org/).

`dCache` is an open-source system, which was developed as well-qualified manager of storing and exchanging of several hundreds of terabytes of data, which is distributed among dozens of disk storage nodes. Despite the fact that location and multiplicity of the data are defined by `dCache` and it's one of the main features of the system based on conguration, CPU load and disk space, the name space is unambiguously represented within a single file system tree. Significant improving of the efficiency of connected tape storage systems provided by `dCache`, by caching and scheduled staging techniques. Additionally, it optimizes the through put to and from data clients as well as smoothly loading of the connected disk storage nodes by dynamically replicating datasets on the detection of loading hot spots. The basic architecture of `dCache` is presented in Figure 1.

`Apache Spark` is an open-source cluster-computing framework, which provides an interface for programming complete clusters with implicit data parallelism. `Apache Spark` has as its architectural foundation the resilient distributed dataset (RDD), a read-only dataset distributed over a cluster of machines, that is maintained in a fault-tolerant way. `Spark` and its RDDs were developed in response to limitations of the `MapReduce` cluster computing paradigm, which forces a particular linear dataflow structure on distributed programs: `MapReduce` programs read input data from disk, map a function across the data, reduce the results of the map, and store reduction results on disk, when the RDD provides the ability to store intermediate results in a distributed memory instead of disk and make the system faster. Differences in structure of paradigms can be found in Figure 2.

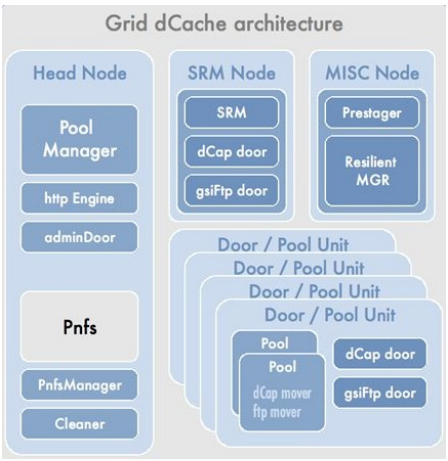

Figure 1: Basic `dCache` architecture

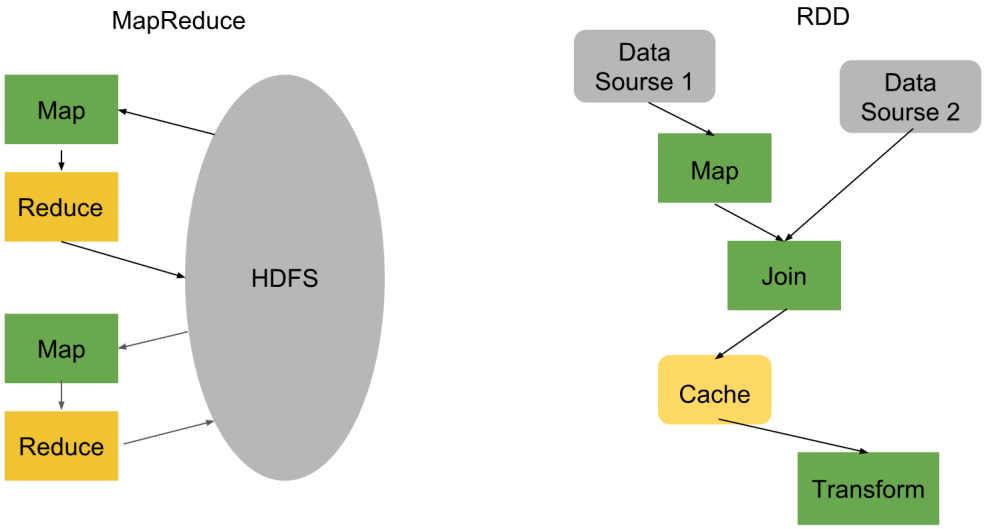

Figure 2: `MapReduce` vs `RDD`

`Spark Core` is the foundation of the whole Spark project. It provides distributed task dispatching, scheduling, and basic I/O functionalities, exposed through an application programming interface centered on the RDD abstraction. This interface represents a higher-order model of programming: a "driver" program invokes parallel operations such as map, filter or reduce on an RDD by passing a function to Spark, which then schedules the function's execution in parallel on the cluster. These operations, and additional ones such as join, take RDD as input and produce new RDDs. RDDs are immutable and their operations are lazy; fault-tolerance is achieved by keeping track of the "lineage" of each RDD so that it can be reconstructed in the case of data loss.

`Spark SQL` is a component on top of `Spark Core`, which provides a data abstraction (Data Frames), which provides support for structured and semi-structured data. `Spark SQL` provides a domain-specific language (DSL) to manipulate DataFrames in `Scala, Java, or Python`. It also provides SQL language support, with command-line interfaces and ODBC/JDBC server.

3 Analysis

First data (example of request event) from billing files was parsed using `Python 3` by logical parts of each event.

Example of request event from billing file:

```
07.01 00:17:42 [door:GFTP-dcache-door-atlas01-AAVv41VGa3g@dcache-door-atlas
01gridftp Domain:request] ["/DC=ch/DC=cern/OU=Organic Units/
OU=Users/CN=atlact1/CN=555105/CN=Robot:  ATLAS aCT 1":
40001:4000:131.169.161.71] [00009E1C9048153B48378E441A4FEB0CDCE1,11409201]
[/pnfs/desy.de/atlas/dq2/atlasdatadisk/rucio/data1713TeV/a6/93/
log.14519703.000163.job.log.tgz.1.rucio.upload] atlas:atlasdatadisk@osm 177
0 0:""
```

The code of parsing pools is represented above:

```
def funcpool(row):
    for r in row:
        if "pool:" in r:
            pool = r.split(":")[1]
        if "@" in pool:
            return pool.split("@")[0]
    return pool
```

Also parsing was made for every other part of the event (Time, Type, PGroup, PnfsId,

Size, Door, Protocol, Domain, ClientIp, Return, ReturnMessage). And after all parsing dataframe (with type 'pyspark.rdd.PipelinedRDD') can be used instead of billing file. The example of parsed event is represented in Table 1.

Time	07.01 00:17:42
Type	request
Pool	
PGroup	atlasscratchdisk
PnfsId	00009E1C9048153B48378E441A4FEB0CDCE1
Size	11409201
Door	GFTP-dcache-door-atlas01-AAVv41VGa3g
Protocol	GFTP
Domain	GFTP-dcache-door-atlas01
ClientIp	131.169.161.71
Return	0
ReturnMessage	

Table 1: Example of result of converting data from billing file

4 Results

4.1 Functioning capacity of the system

By ATLAS data

For checking of functioning capacity of the system, all transfers in ATLAS group were counted by day in 2017 (Figure 3) and also in the month average (Figure 4). In Figure 3 white color also indicates internal transfers between DESY nodes.
From figures you can see some high points in May and June due to conferences. The majority of these points were initiated by clients outside of DESY. In June, July and October high points are triggered by DESY only. During other months the value of transfers is small compared to the peaks, so probably the power of the system should be increased in months or weeks, which are followed by some sessions, and some energy could be saved in periods with not high activity. Turning off the power of the machines in low periods and asking any third party for the enough and so high (as usual) power can be one of the possible solutions. Accordingly, it can save money for the whole project.

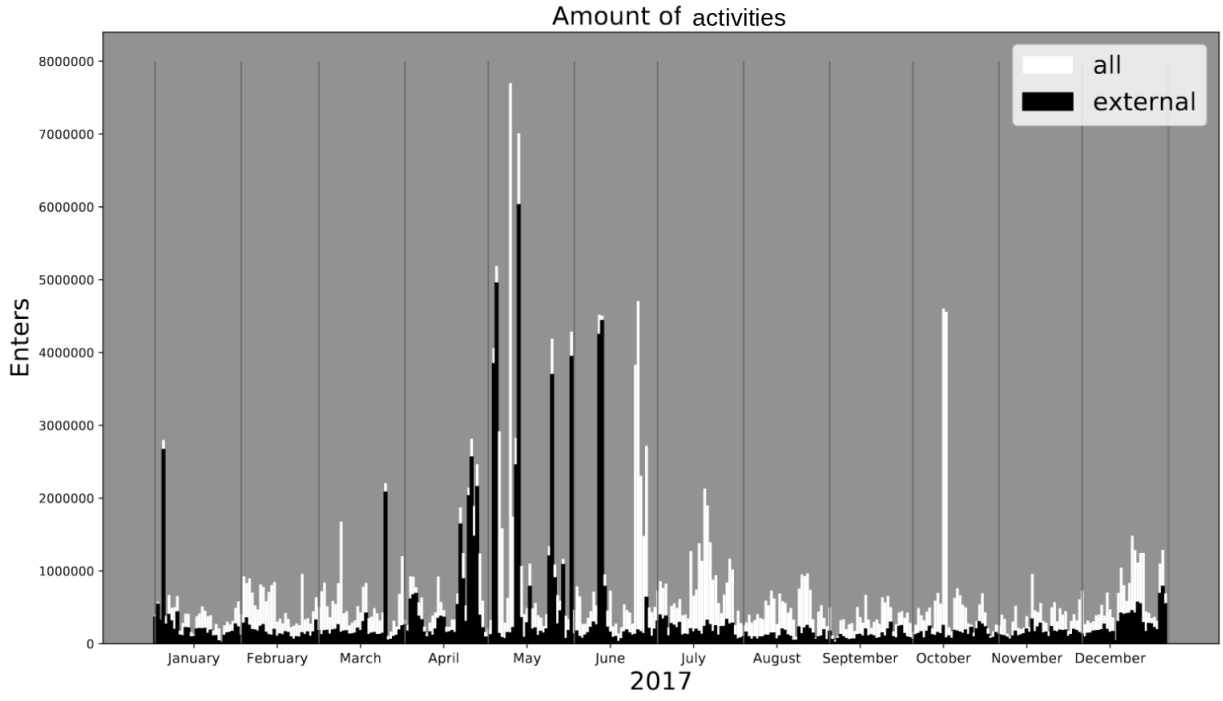

Figure 3: Transfers for the whole year

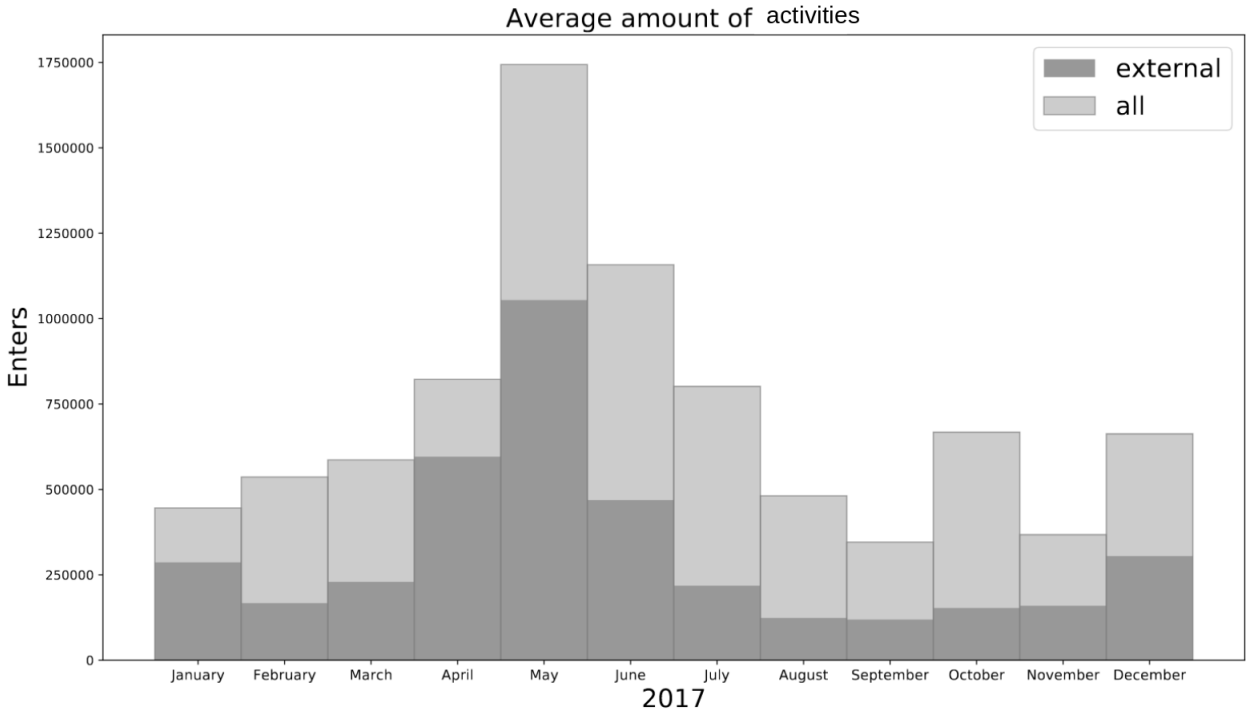

Figure 4: Average amount of transfers by months

By CMS data

In Figure 5 amount of transferred data can be seen by days with scale in petabytes (PB). There are the high peaks in the middle of January and February, in the end of June and August also.

In Figure 6 the value of error 451 is represented and significant points can be found in April, May, July, and August especially. In other time amount of errors not so high, so a good point is to find out why or by which users these errors were initiated to prevent such situations in future.

In Figure 7 the value of error 666 is located. By whole period amount of errors 666 is higher then amount of error 451 in Figure 6. But hot peaks are made in the same time. And also these peaks are connected with Figure 5 in peaks with transferred data with value around 20 PB.

Next years in periods with the high amount of transferred data and errors IT department should pay attention to the status and the stability of the system. Also a research about the reason of amount of errors can be made.

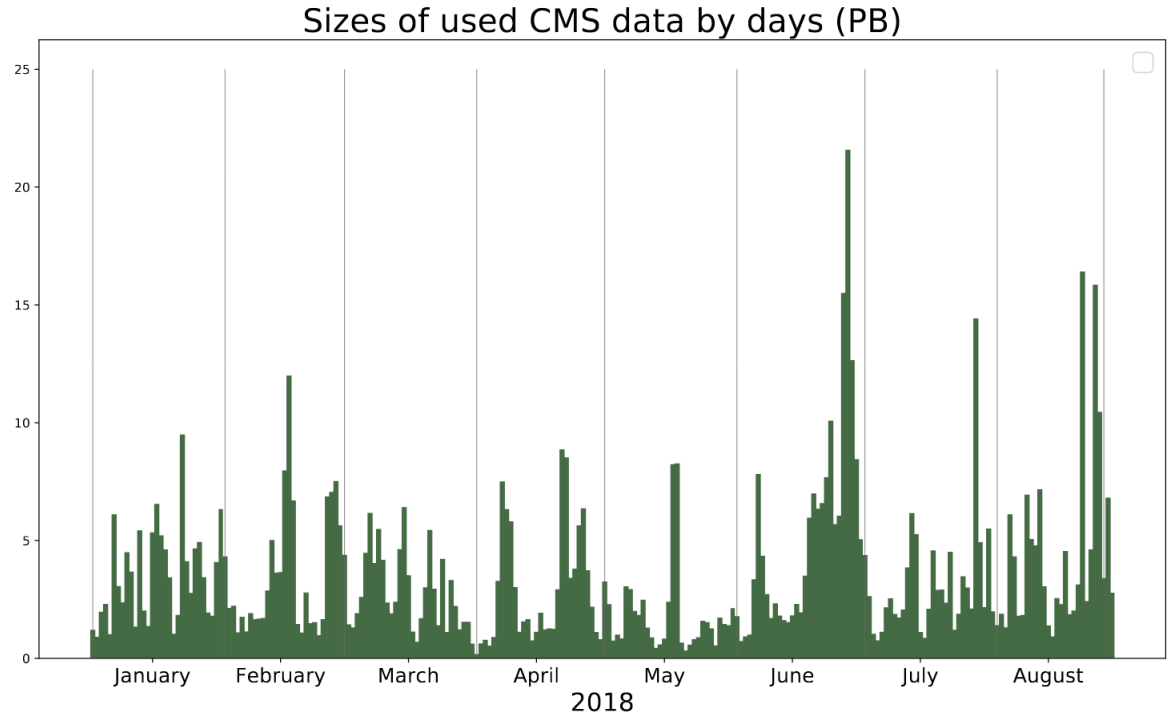

Figure 5: Value of transferred data by CMS group in petabytes by day during the period January-August, 2018

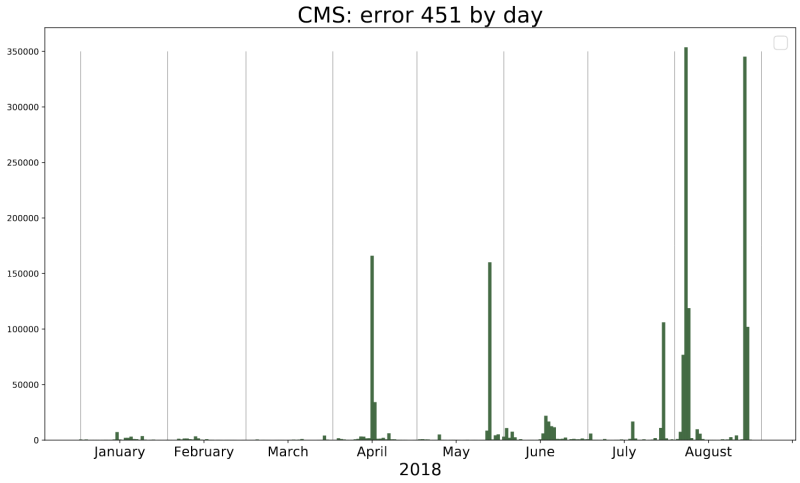

Figure 6: Amount of error 451 in CMS group by day in January-August, 2018

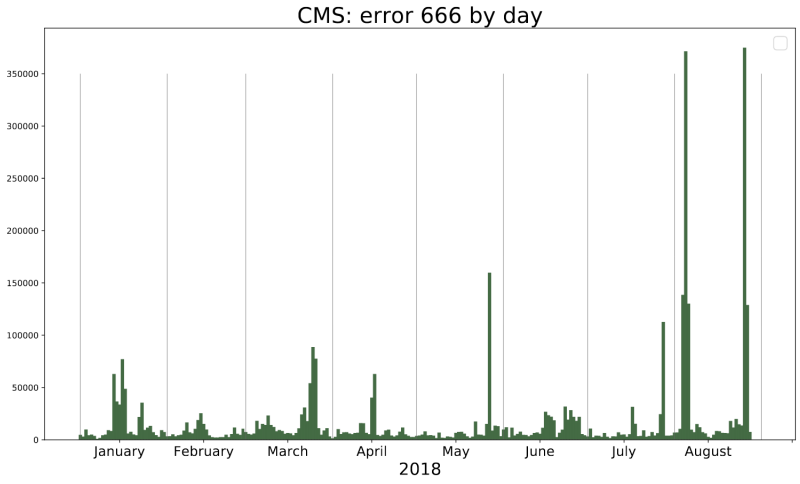

Figure 7: Amount of error 666 in CMS group by day in January-August, 2018

4.2 Request issues

The analysis of the data for August, 2018 has represented that sometimes failed request (with 451 error) follows the successful transfer, which is not the proper way of the system to work.

ATLAS		CMS	
In DESY	Outside DESY	In DESY	Outside DESY
1221	117	12761	886

Table 2: Amount of error 451 in August, 2018 by working groups ATLAS and CMS

From the above Table 2 we can see that most of all such issues are producing by users inside the DESY network. By checking the Door it turned out that:

- some of creators is grid-mon1.desy.de (19 times for ATLAS and 9 for CMS)
- for ATLAS other creators are different batches
- other errors in CMS are produced only by batch0405.desy.de (12752 times per month)

Probably it's necessary to find out what was the reason of these errors, and if it's not about IT administration, connect to most problematic users.

5 Conclusion

During the DESY Summer Student Program 2018 , the analysis was made and the basic billing files can be seen as not applicable by themselves, so first they have to be converted with Apache Spark. Then using SQL data can be taken for different periods of time, different points of view, etc., demonstrative plots can be made with the libraries of Python and next research of the result can be done for figuring out the curious peaks, etc.

6 References

1. dCache online-page: https://www.dcache.org/
2. Apache Spark documentation: https://spark.apache.org/docs/latest/rdd-programming-guide.html
3. Python documentation: https://docs.python.org/3/
4. Matplotlib documentation: https://matplotlib.org/