Analysing Content of large dCache Databases

Using Apache Spark to Improve Queries to dCache Databases

Christian Voß
DESY

HELMHOLTZ

Comparison between DESY-Hamburg dCache Installations

Use dCache for Variety of Customers

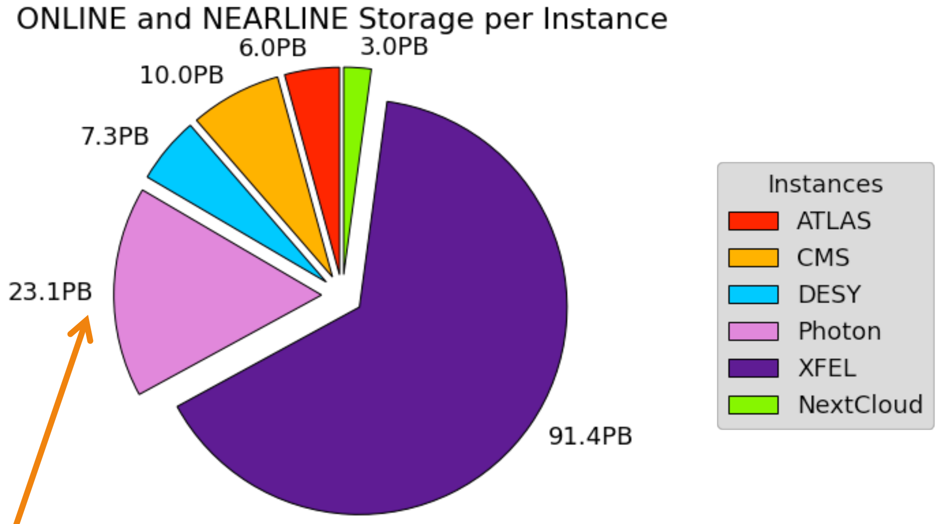
Particle Physics

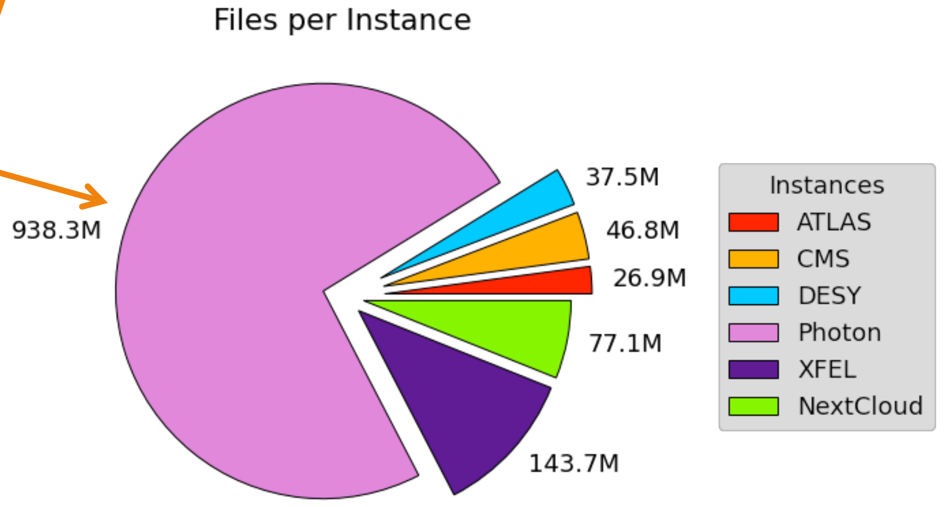
- DESY is a Tier-2 centre for ATLAS, CMS and LHCb
- Raw data centre for Belle II
- Tier-0 for ILC
- Host small on-site HEP communities

Research with Photons

- DESY hosts several on-site photon sources
- DESY hosts data for European XFEL
- Data analysis done on HPC w/ GPFS
- Role of dCache: archival of RAW data
- Second order analyses

Write
many, small files





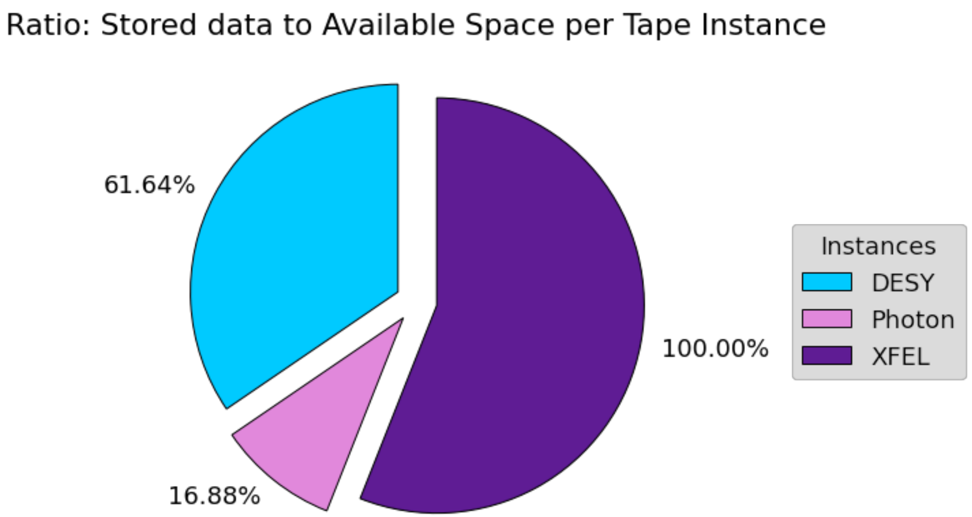
Focus Today: Dealing with Photon Science Instances

dCache Operational Cross-Checks with Tape Library

Provide Tape Storage on

- DESY for Belle II, ILC, IT services
- Photon for PETRA 3, FLASH, CFEL, CSSB, and Machine Group
- XFEL for their raw/calibration/processed data

- Only European XFEL provides enough storage to store on disk and tape



Data Policies

- DESY: single tape copy
- XFEL: two copies on two independent systems, i.e. dCache disk and tape
- PETRA 3/FLASH: two copies on two different tape media

Monitor and Ensure Policies

How to Ensure the File Policies

How to monitor Entries in `t_locationinfo` and in the HSM

- Check locality via SRM:

```
[vossc@naf-it01] ~ $ srmls -l srm://dcache-se-dot.desy.de:8443/pnfs/desy.de/dot/cta-dev/st.99.0 | awk -vFS=: 'NF==1{f=$0}/locality/{print f,$0}'
 42582016 /pnfs/desy.de/dot/cta-dev/st.99.0   locality:NEARLINE
```

- How reliable is Locality: entry in `t_locationinfo` and successful store
- dCache has no knowledge of quality of tape system, only the `uri` is known

```
[chimera=# select ilocation from t_locationinfo where inumber in (select inumber from t_inodes where ipnfsid='0000B09316B5A1FF444A9E93EBD17024698F');
                                 ilocation
--------------------------------------------------------------------------
 osm://osm/?store=petra3&group=p06&bfid=021169072174.605a9c340be65f.413c11
(1 row)
```

- HSM system usually keeping track of its inventory separately

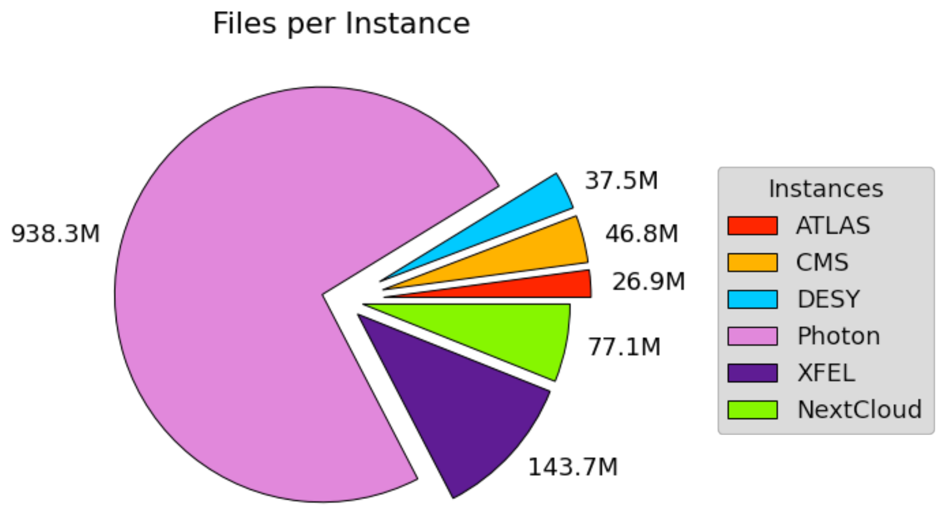
```
[osmdb=> select bfid, volid, status, creationtime from t_location where bfid='021169072174.605a9c340be65f.413c11';
               bfid                | volid | status |         creationtime
-----------------------------------+-------+--------+-------------------------------
 021169072174.605a9c340be65f.413c11 |  3735 |      1 | 2021-03-24 02:56:04.779946+01
 021169072174.605a9c340be65f.413c11 |  4032 |      1 | 2021-03-24 04:45:03.327134+01
 021169072174.605a9c340be65f.413c11 |  5674 |      1 | 2021-12-25 05:12:02.757532+01
(3 rows)
```

- Cross-checks to be done manually → DB dumps and scripts to compare

Limitations of Manual Checks

Extended Query Times for Large Databases

- Running external scripts on dumps can be tedious
- Running against a replica requires a lot of storage
- Queries typically not optimal
- Limited by size of the databases (for us 2.1TiB currently)



Photon dCache

- Last summer → had to make a major scan of dCache + tape archive
- Tape archival queries finished in a few minutes
- Gave up on dCache queries after many hours → encountered a death spiral
  - Query ran into backup period inducing further delays
  - Backup delayed ingest inducing further delays on scan query

Simple Example: Check if all Stored Files are known to HSM

Compare dCache with OSM for Photon dCache

- Remember `OSMTemplate` tag with entry `StoreName`: `petra3:p11@osm` maps directly into store in OSM
- Multiple stores in our instances → need to filter these

dCache

- Filter on `t_storageinfo` for specific store
- Find `inumber`, i.e. files, for a given store
- Find all entries in `t_locationinfo` for selected range for `inumber` that matches the HSM
- Query whole DB, no profit from cached data
- Worse → invalidate parts of the file system cache
- Write into output format (parse content of `t_locationinfo` to extract `bfid`)

OSM

- Select corresponding store namespace in database
- Filter on `bfid` in location table read from dCache export
- Find associated tapes
- Merge into output list

A more Convenient Alternative

Using Table Joins and Memory to Store Data

- Looking for something more convenient than comparing output files → e.g. table joins
  → Want a simple query for files without valid HSM location

Files with no bfid: 2

inumber	clean_bfid	ibfid	store	sGroup	ictime
23851963	null	021169072174.6059e43707622c.0df824	desy	dot-test	2021-03-23 14:07:54.393
24018973	null	021169072174.6066342f00261a.5aa3b8	desy	dot-test	2021-04-01 23:00:02.023

OSM knows nothing about this file

dCache entry for tape location

- Avoid interference with production database
  - Needs space to save dumps and restore
  - Usually use a separate host for that: Our Limit: Find 3TB of performing disk space on a node

→ Can exercise the checks in memory

How to Combine/Overcome both Join and Space Limitation

Do Joined Searches in Memory with Apache Spark Cluster

- Checking our largest DB made us desperate
- Use Spark for billing/logging analyses since 2018
- Initial import large data set time intensive
- All further queries are fast/interactive

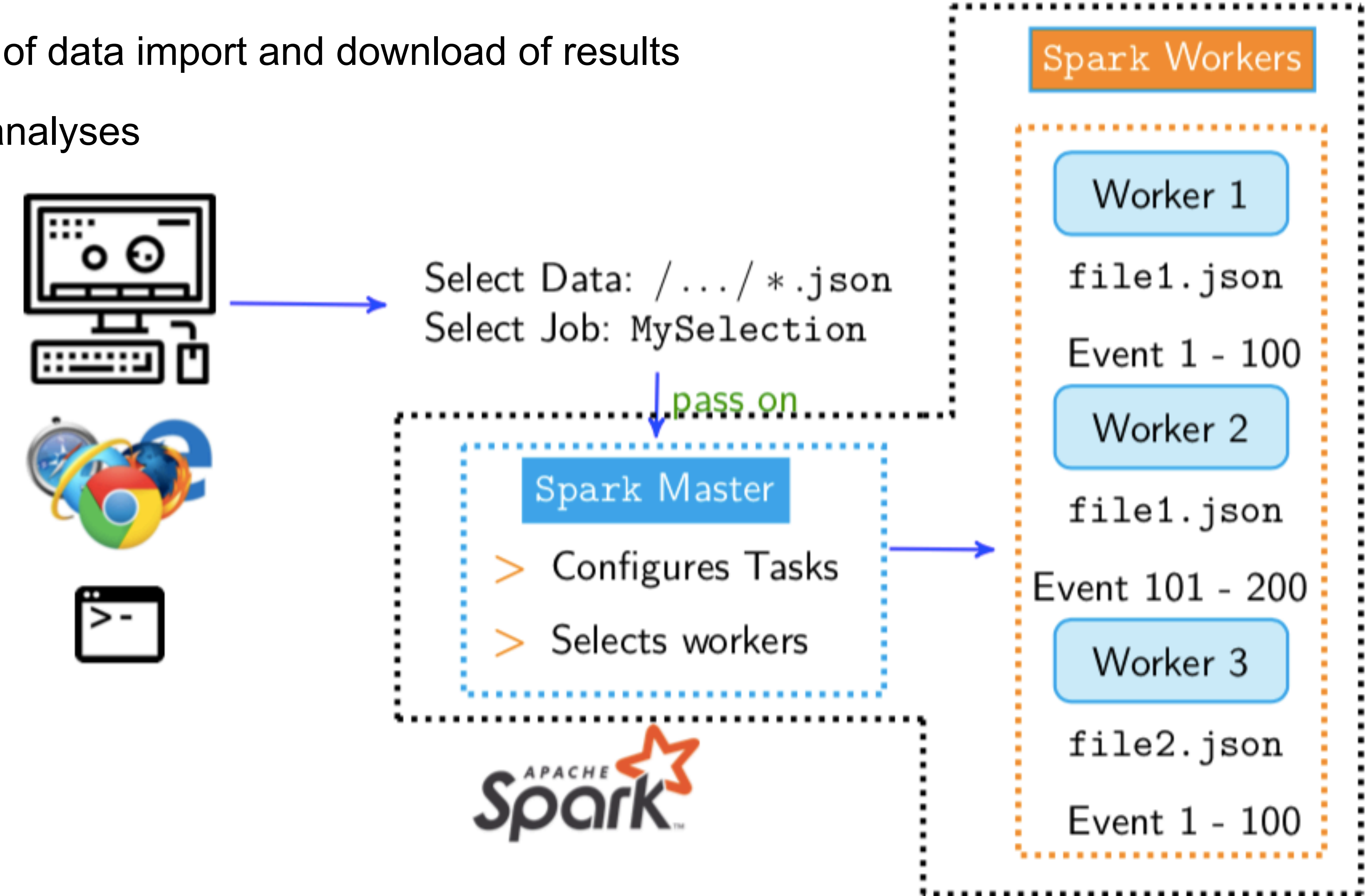
7	collect at <ipython-input-51-8d304014346c>:1 collect at <ipython-input-51-8d304014346c>:1	2019/05/18 20:09:24	2 s	1/1 (2 skipped)	200/200 (24603 skipped)
6	collect at <ipython-input-17-b23218fa0fed>:1 collect at <ipython-input-17-b23218fa0fed>:1	2019/05/18 19:46:28	9 s	2/2 (1 skipped)	400/400 (24403 skipped)
5	javaToPython at NativeMethodAccessorImpl.java:0 javaToPython at NativeMethodAccessorImpl.java:0	2019/05/18 19:46:11	17 s	2/2	24603/24603
4	count at NativeMethodAccessorImpl.java:0 count at NativeMethodAccessorImpl.java:0	2019/05/18 19:42:50	12 s	3/3 (1 skipped)	401/401 (24403 skipped)
3	count at NativeMethodAccessorImpl.java:0 count at NativeMethodAccessorImpl.java:0	2019/05/18 19:42:27	23 s	2/2	24603/24603
2	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2019/05/18 19:38:30	32 s	2/2	24603/24603
1	count at NativeMethodAccessorImpl.java:0 count at NativeMethodAccessorImpl.java:0	2019/05/18 18:35:46	58 min	2/2	24404/24404
0	count at NativeMethodAccessorImpl.java:0 count at NativeMethodAccessorImpl.java:0	2019/05/18 17:35:34	58 min	2/2	24404/24404

- Can you Spark for our database analysis as well?

Apache Spark Cluster

Very Short Overview

- Spark consists of a scheduler and a number of workers
- Spark application controls the flow of data import and download of results
- Primary usage: large scale billing analyses



- Usually run on batch-nodes
- Import billing data via dCache using NFS
- DB analysis with different needs
  - Allow access to DB
  - Limit number clients and connections

Connect Spark with PostgreSQL

Three Major Questions

1. Resources to import a database with ~500GiB of table size
   - Quite straight forward: use decommissioned dCache heads, while old CPUs they have enough memory and are not in use
2. How to make Spark aware of the database format
   - Use the regular Java driver for PostgreSQL
   - Expose the jar to Spark/make Spark aware of the driver

```
Singularity> pwd
/opt/spark/jars
Singularity> ls postgresql-42.2.22.jar
postgresql-42.2.22.jar
```

3. How to instruct Spark to split the workload among workers
   - Make Spark use the PostgreSQL driver
   - Configure connection
   - Configure the worker payload

   - Six worker
   - Split/partition by default index `inumber`
   - Observe six distinct connections with selections based on `inumbers`

```
t_inodes = sqlContext.read.format("jdbc")\
  .option("driver", "org.postgresql.Driver")\
  .option("url", "jdbc:postgresql://{}/chimera".format(db_host))\
  .option("dbtable", "t_inodes")\
  .option("numPartitions",6)\
  .option("partitionColumn", "inumber")\
  .option("lowerBound", low_inumber)\
  .option("upperBound",up_inumber)\
  .load()
```

Modifications to dCache database

Extract the ID for the Tape File from its Location

- Internal index and nearline location in form of an URI, nearline system keeps its internal ID

```
 inumber  |                                 ilocation
-----------+--------------------------------------------------------------------------------
 896311558 | osm://osm/?store=petra3&group=p06&bfid=021169072174.605a9c340be65f.413c11
```

```
                bfid                | slid | volid
------------------------------------+------+-------
 021169072174.605a9c340be65f.413c11 |   96 |  3735
 021169072174.605a9c340be65f.413c11 |   69 |  4032
 021169072174.605a9c340be65f.413c11 |  248 |  5674
```

→ Make use of internal functions provided by Spark

```
split_location = pyspark_function.split(t_locationinfo.ilocation, '[&=]')
```

```
df_location = t_locationinfo.select("inumber","ilocation", "ictime", split_location.getItem(1).alias('istore'),\
                                    split_location.getItem(3).alias('sGroup'),\
                                    split_location.getItem(5).alias('ibfid')).where("ilocation like 'osm://%'")
df_location.show(5,True)
```

```
+---------+--------------------+--------------------+------+---------+--------------------+
|  inumber|           ilocation|              ictime|istore|   sGroup|               ibfid|
+---------+--------------------+--------------------+------+---------+--------------------+
|       12|osm://osm/?store=...|2014-12-05 21:06:...|   ttf|  ttf2-13|00144fa0d390.5482...|
|      135|osm://osm/?store=...|2014-11-28 22:19:...|   ttf|  ttf2-13|00144fa0d390.5478...|
|      168|osm://osm/?store=...|2012-10-11 19:17:...|   ttf|  ttf2-12|00144fa0d390.5076...|
|      177|osm://osm/?store=...|2012-01-30 01:50:...|   ttf|  ttf2-12|00144fa0d390.4f25...|
|318411187|osm://osm/?store=...|2017-09-02 05:22:...|   ttf|ttf2-17-d|021169072174.59aa...|
+---------+--------------------+--------------------+------+---------+--------------------+
only showing top 5 rows
```

Linking up with the Tape Database

Perform the Join

- After modification → two tables each with a unique identifier on which to join

```
chimera_osm_join = df_dCache.join(df_osm, df_osm.clean_bfid == df_dCache.ibfid,how='fullouter')
```

```
chimera_osm_join.printSchema()
chimera_osm_join.cache()
```

```
root
 |-- inumber: long (nullable = true)
 |-- ilocation: string (nullable = true)
 |-- ictime: timestamp (nullable = true)
 |-- istore: string (nullable = true)
 |-- sGroup: string (nullable = true)
 |-- ibfid: string (nullable = true)
 |-- pnfsid: string (nullable = true)
 |-- filesize: long (nullable = true)
 |-- clean_bfid: string (nullable = true)
```

- Access to full table showing both information

```
columns = ['inumber', 'clean_bfid', 'ibfid', 'istore', 'sGroup', 'ictime']
chimera_osm_join.select(*columns).show(10,False)
```

```
+---------+------------------------------------+------------------------------------+------+-------+--------------------------+
|inumber  |clean_bfid                          |ibfid                               |istore|sGroup |ictime                    |
+---------+------------------------------------+------------------------------------+------+-------+--------------------------+
|151428011|0003ba08dd9e.45e66ffe09086a.a35a1f|0003ba08dd9e.45e66ffe09086a.a35a1f|ttf   |ttf2-07|2010-01-07 07:49:51.84876|
|151428649|0003ba08dd9e.45ef3d610d3d16.9d9529|0003ba08dd9e.45ef3d610d3d16.9d9529|ttf   |ttf2-07|2010-01-07 07:49:51.84876|
```

Performing the Check

Or Let's Find Lost Data

- Have full join → filter for `null` entries

```
df_null = chimera_osm_join.select(*columns).where("clean_bfid is null AND ibfid!='*'")
print("Files with no bfid: {}".format(df_null.count()))
df_null.show(10,False)
```

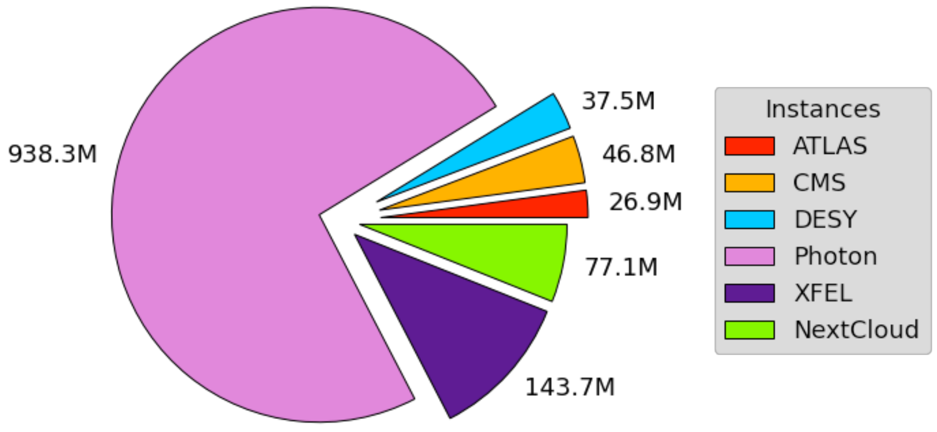
```
Files with no bfid: 3
+---------+----------+----------------------------------+------+----------+----------------------+
|inumber  |clean_bfid|ibfid                             |istore|sGroup    |ictime                |
+---------+----------+----------------------------------+------+----------+----------------------+
|82693879 |null      |021169072094.55d1f0820b1b29.bf541d|desy  |dot-test  |2021-08-17 16:40:08.769|
|6638999  |null      |021169072094.55d1fdfa09c923.3deefe|desy  |dot-test  |2021-08-17 17:42:10.382|
|69730326 |null      |021169072094.55d20a1c082766.2bea6d|desy  |dot-test  |2021-08-17 18:28:08.485|
+---------+----------+----------------------------------+------+----------+----------------------+
```

- Find all 'lost' files → files kown to dCache but not the HSM
- Find all HSM orphans → file long since deleted on dCache

Need for Speed – What is the Advantage

Or is it Being German → 1M$ solution for 10$ Problem

Files per Instance



- Advantage: no need for dedicated storage
  → RAM/CPUs usually easy to find
- Comes down to scale → usually dump/import/join usually fine
  - At worst dump/restore will take 1h (~50M files)
  - Remember: Photon DB at DESY has 1000M files
- What is the actual time scale? → significant decrease in latency
  → overall faster checks

Completed Stages (9)

Stage Id	Pool Name	Description		Submitted	Duration	Tasks: Succeeded/Total
8	default	collect at <ipython-input-5-1f08d7c6491c>:126	+details	2022/05/16 20:20:05	15 s	6/6
7	default	count at NativeMethodAccessorImpl.java:0	+details	2022/05/16 18:57:31	0.2 s	1/1
6	default	count at NativeMethodAccessorImpl.java:0	+details	2022/05/16 15:58:57	3.0 h	6/6
5	default	count at NativeMethodAccessorImpl.java:0	+details	2022/05/16 15:58:57	0.2 s	1/1
4	default	count at NativeMethodAccessorImpl.java:0	+details	2022/05/16 07:42:36	8.3 h	6/6
3	default	count at NativeMethodAccessorImpl.java:0	+details	2022/05/16 07:42:36	97 ms	1/1
2	default	count at NativeMethodAccessorImpl.java:0	+details	2022/05/16 07:42:32	3 s	200/200
1	default	count at NativeMethodAccessorImpl.java:0	+details	2022/05/16 07:42:26	6 s	1/1
0	default	count at NativeMethodAccessorImpl.java:0	+details	2022/05/16 07:42:26	2 s	1/1

Expensive Query

dCache Import

HSM Import

Need for Speed – Is Speed-Up consistent?

Running Queries versus Database in Memory (Using Spark's Caching Capabilities)

- One single query finishes faster overall → what about consecutive queries?

Execution time constant

Made more than 10k queries since import

Completed Stages (49162, only showing 922)

Page: 1 2 3 4 5 6 7 8 9 10 >   10 Pages. Jump to 1

Stage Id	Pool Name	Description ▲		Duration	Tasks: Succeeded/Total	Input
121630	default	collect at <ipython-input-5-1f08d7c6491c>:126 +details	2022/05/24 22:48:29	15 s	6/6	4.1 GB
121606	default	collect at <ipython-input-5-1f08d7c6491c>:126 +details	2022/05/24 22:47:40	15 s	6/6	4.1 GB
121582	default	collect at <ipython-input-5-1f08d7c6491c>:126 +details	2022/05/24 22:46:47	15 s	6/6	4.1 GB
121558	default	collect at <ipython-input-5-1f08d7c6491c>:126 +details	2022/05/24 22:45:56	16 s	6/6	4.1 GB
121534	default	collect at <ipython-input-5-1f08d7c6491c>:126 +details	2022/05/24 22:45:03	14 s	6/6	4.1 GB
121510	default	collect at <ipython-input-5-1f08d7c6491c>:126 +details	2022/05/24 22:44:11	15 s	6/6	4.1 GB

- Lifetime → how stable is the Spark import

Active for days, longer than the import is valid

Running Applications (1)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20220520092427-0000 (kill)	dot-postgres-analysis	180	120.0 GB	2022/05/20 09:24:27	vossc	RUNNING	119.5 h

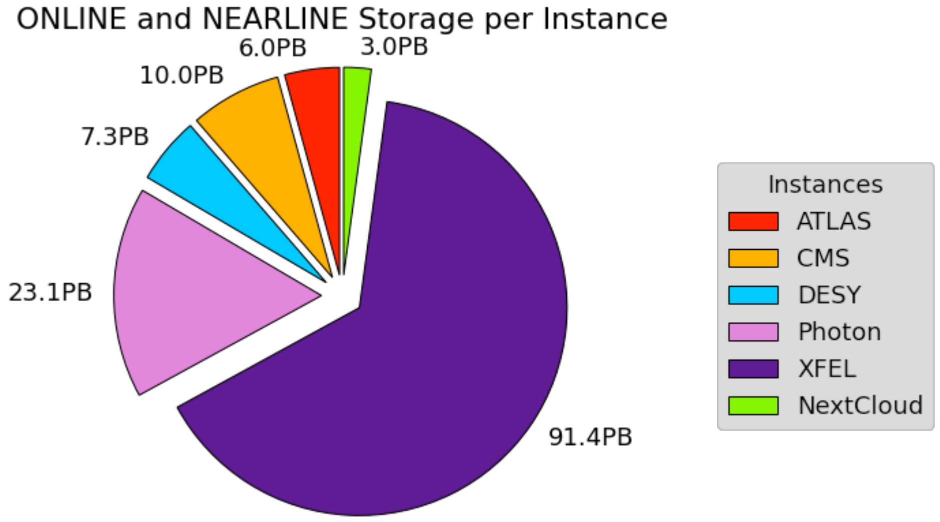
Back to Photon dCache – Production Example

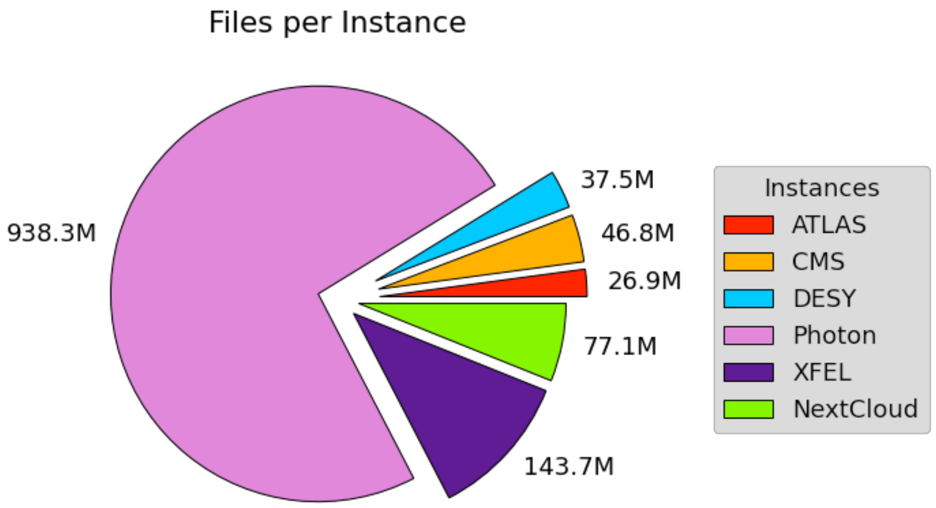
Using Spark to Check for Second Copy on Tape before Removal

- Follow up to our studies from Summer
- Users write many small files supposed to be stored on tape
- Make use of the Small File Plugin/HSM

Locality Check for a given beam-time

- Truth is on the offline cluster for the photon sources → their paths
- file packed → container flushed → tape system creates 2nd copy
- Import `t_locationinfo` → create table in Spark for small-files and tape-files based on URI → cache in Spark
- Import of tape `t_location` → create a joined table based in file-ids for all files in dCache and HSM → cache in spark
- Select all container files based on `inumbers` of the small files
- Store output → container and tapes for each file → trigger delete





Summary

Using Spark for Checks Based on Database Queries

- Adapted Apache Spark to help us run (and finish) checks between dCache and our tape system
- Eliminates the need for additional large DB nodes (major benefit for us)
- Lucky: dCache, OSM/CTA use PostgreSQL
- Spark should be flexible enough to use other Java DB drivers as well
- Comparison with TSM → use a dump to CSV and import into Spark

- Global checks can be done quickly and easily
- Checks based on path need more effort:
  - Limiting factor is now the namespace lookup → Spark cannot import PostgreSQL functions (`path2inumber`)
  - Still use direct queries to Chimera → need to improve skills with Spark to analyse `t_dirs` recursively

- Take the last step → de-Jupyter-ise the checks to run on dedicated clusters

Thank you