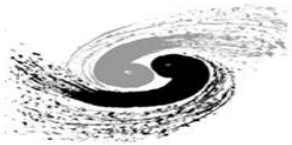


中國科學院高能物理研究所
Institute of High Energy Physics
Chinese Academy of Sciences

高能所計算中心
IHEP Computing Center

# Anomaly detection of I/O behaviors in HEP computing cluster based on unsupervised machine learning

**Lu Wang (wanglu@ihep.ac.cn)**

**Computing Center, Institute of High Energy Physics, CAS**

**2021-11-29**

# Outline

# Problematic I/O patterns



- Problematic I/O is a major cause of low efficiency HEP jobs.
  - Large distance seeks in a large file + small read (<4KB) , small writes + sync in parallel …
  - disable the optimizations of distributed storage systems
    - client cache, read ahead, write back etc.,
  - Sometimes these usages are unavoidable, but we need to detect them in time and control their impacts on the overall efficiency
- Traditional diagnosis of these patterns involves monitoring of storage servers, alarming of heavy workloads and backtracking of the workloads…
  - Big MTTR (Mean Time to Restoration ), typically in unit of hours
- We implemented a direct, instant and automatic way of problematic I/O detection based on unsupervised machine learning.

# I/O pattern of a HEP job(1/2)



- I/O pattern Includes operations of both data and metadata
  - described by histograms of size, bandwidth and frequency
- Lustre file system provides a way to attach job identity with I/O requests sending to servers
  - The identity can be process name, client node name, process id or any ID which can be read from a process's environment variables
  - We use HT-Condor job ID as job identity

    lctl set_param jobid_var=_CONDOR_IHEP_JOB_ID
  - On login nodes without HT-Condor ID, Lustre will use the default identity as "procname+uid"
- Lustre servers make statistics of running jobs' IO patterns in memory, and histograms can be queried by "/proc" interfaces

# I/O pattern of a HEP job(2/2)



## ●On Metadata server

```
[redacted]# cat /proc/fs/lustre/mdt/lhaasofs-MDT0000/job_stats |grep job_id |wc -l
1529
[redacted]# cat /proc/fs/lustre/mdt/lhaasofs-MDT0000/job_stats |head -n 22
job_stats:
- job_id:          lfs.0
  snapshot_time:   1638074772
  open:            { samples:           0, unit:  reqs }
  close:           { samples:           0, unit:  reqs }
  mknod:           { samples:           0, unit:  reqs }
  link:            { samples:           0, unit:  reqs }
  unlink:          { samples:           0, unit:  reqs }
  mkdir:           { samples:           0, unit:  reqs }
  rmdir:           { samples:           0, unit:  reqs }
  rename:          { samples:           0, unit:  reqs }
  getattr:         { samples:          64, unit:  reqs }
  setattr:         { samples:           0, unit:  reqs }
  getxattr:        { samples:           0, unit:  reqs }
  setxattr:        { samples:           0, unit:  reqs }
  statfs:          { samples:        7234, unit:  reqs }
  sync:            { samples:           0, unit:  reqs }
  samedir_rename:  { samples:           0, unit:  reqs }
  crossdir_rename: { samples:           0, unit:  reqs }
  read_bytes:      { samples:           0, unit:  reqs, min:       0, max:       0, sum:
             0 }
  write_bytes:     { samples:           0, unit:  reqs, min:       0, max:       0, sum:
             0 }
  punch:           { samples:           0, unit:  reqs }
```

- Total running jobs accessing this device: 1529
- Number of metadata operation: 22

## ●On data server

```
[redacted]# cat /proc/fs/lustre/obdfilter/lhaasofs-OST0000/job_stats |grep job_id|w
c -l
488
[redacted]# cat /proc/fs/lustre/obdfilter/lhaasofs-OST0000/job_stats |head -n 15
job_stats:
- job_id:          lfs.0
  snapshot_time:   1638075012
  read_bytes:      { samples:           0, unit: bytes, min:       0, max:       0, sum:
             0 }
  write_bytes:     { samples:           0, unit: bytes, min:       0, max:       0, sum:
             0 }
  getattr:         { samples:           0, unit:  reqs }
  setattr:         { samples:           0, unit:  reqs }
  punch:           { samples:           0, unit:  reqs }
  sync:            { samples:           0, unit:  reqs }
  destroy:         { samples:           0, unit:  reqs }
  create:          { samples:           0, unit:  reqs }
  statfs:          { samples:        7236, unit:  reqs }
  get_info:        { samples:           0, unit:  reqs }
  set_info:        { samples:           0, unit:  reqs }
  quotactl:        { samples:           0, unit:  reqs }
```

- Total running jobs accessing this device: 488
- Number of data operations: 15

# Why unsupervised machine learning ?



- With above I/O patterns, we can simply make detection by heuristic policies
  - The threshold of problematic job for each operation is difficult to determine
  - can not adapt to changes of hardware and software
  - can not detect new  anomalies
- Machine learning provides a data driven way to do this task
- In this case, we used a ML algorithm called Isolation Forest
  - Unsupervised, do not need tagged data set for model training
  - linear time complexity, low memory requirement, works well in high dimensional problems that have a large number of irrelevant attributes, and in situations where training sets do not contain any anomalies

# Outline

# Data collection



- On IHEP cluster there are 19 Lustre metadata servers, 135 Lustre data servers
- Lustre provides an open sourced collectD plugin for collection of performance metrics, which include those jobstats
  - GitHub - LiXi-storage/barreleye: Lustre Monitoring System
- It is both possible for the plugin to in store these metrics on local disks or export to remote data collectors
  - Local disk file system will run out of inode soon with billions of jobs multiplied by tens of metrics
  - we export collectD results to our central ElasticSearch Database

# Data Query



- Data items in ElasticSearch DB

| subdisk | trigger | value | timev | operation | fsname |
|---|---|---|---|---|---|
| 0 | OST0000 | 122951105_0 | 1.100000 | 1617465627 | write_samples | lhaasofs |
| 1 | OST0000 | 122951129_0 | 68.266667 | 1617465627 | min_read_bytes | lhaasofs |
| 3 | OST0000 | 123660742_0 | 68.266667 | 1617465627 | max_read_bytes | lhaasofs |
| 27 | OST0006 | rsync_12142 | 4437.333333 | 1617465715 | max_read_bytes | lhaasofs |
| 28 | OST0006 | rsync_12142 | 0.750000 | 1617465715 | setattr lhaasofs | |
| 45 | OST000b | 123647284_0 | 1.050000 | 1617465715 | write_samples | lhaasofs |
| 54 | OST000b | rsync_12142 | 68.266667 | 1617465715 | min_read_bytes | lhaasofs |

```
# grep "job" 2021-05-09_2021-05-10_*
2021-05-09_2021-05-10_besfs4_gensample.log:Unique jobs:24421
2021-05-09_2021-05-10_besfs4_gensample.log:Average samples per job:248
2021-05-09_2021-05-10_besfs5_gensample.log:Unique jobs:73305
2021-05-09_2021-05-10_besfs5_gensample.log:Average samples per job:36
2021-05-09_2021-05-10_besfs6_gensample.log:Unique jobs:4825
2021-05-09_2021-05-10_besfs6_gensample.log:Average samples per job:49
2021-05-09_2021-05-10_bprofs_gensample.log:Unique jobs:30516
2021-05-09_2021-05-10_bprofs_gensample.log:Average samples per job:71
2021-05-09_2021-05-10_lhaasofs_gensample.log:Unique jobs:241475
2021-05-09_2021-05-10_lhaasofs_gensample.log:Average samples per job:13
```

- With query results of certain "*timestamp range, fsname and device name*" we can build data samples sorted by job_ID

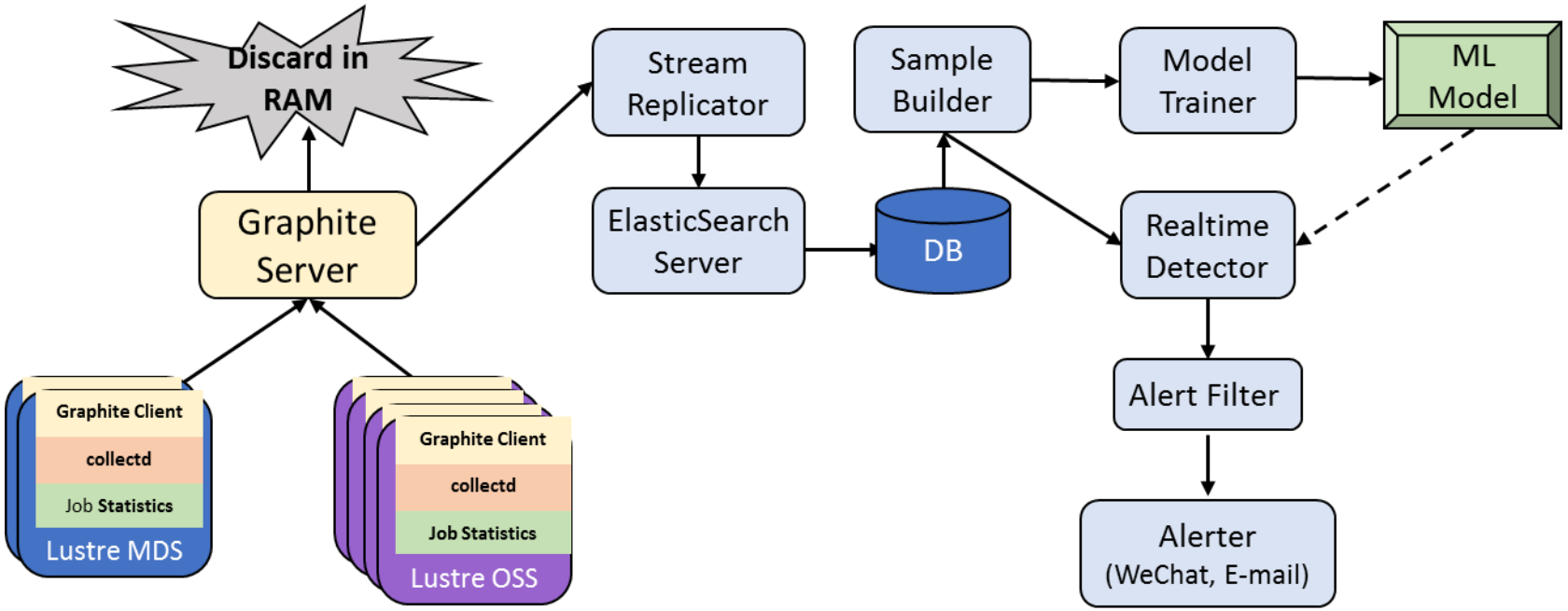
```
# grep "samples gen" 2021-05-09_2021-05-10_*
2021-05-09_2021-05-10_besfs4_gensample.log:Total samples generated:6074420
2021-05-09_2021-05-10_besfs5_gensample.log:Total samples generated:2668111
2021-05-09_2021-05-10_besfs6_gensample.log:Total samples generated:238760
2021-05-09_2021-05-10_bprofs_gensample.log:Total samples generated:2194122
2021-05-09_2021-05-10_lhaasofs_gensample.log:Total samples generated:3198088
```

```
# grep "Query" 2021-05-09_2021-05-10_*
2021-05-09_2021-05-10_besfs4_gensample.log:Query used: 1972.941382 seconds
2021-05-09_2021-05-10_besfs5_gensample.log:Query used: 1223.382304 seconds
2021-05-09_2021-05-10_besfs6_gensample.log:Query used: 705.211348 seconds
2021-05-09_2021-05-10_bprofs_gensample.log:Query used: 1013.886942 seconds
2021-05-09_2021-05-10_lhaasofs_gensample.log:Query used: 1364.852332 seconds
```

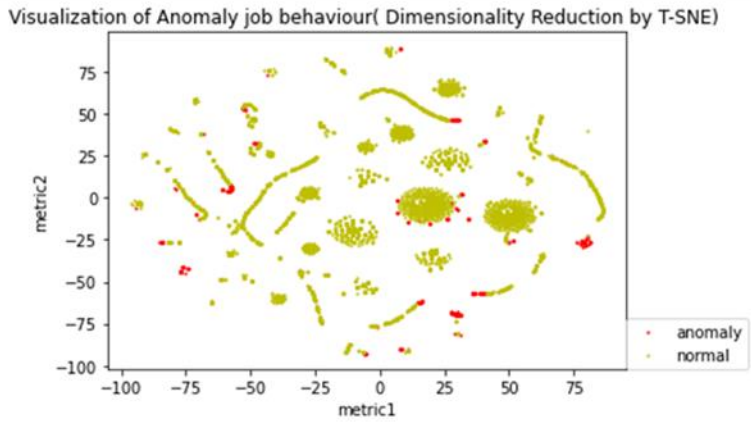
- Since May 2021, We have collected 3.2 billion samples of data operation, 0.9 billion samples of metadata operation

# Outline

# Overall design




```
ostoplist = ["read_samples", "write_samples", "destroy",
             "create","get_info", "set_info", "quotactl",
             "sum_read_bytes","sum_write_bytes"]
mdtoplist = ["open", "close", "mknod", "link", "unlink",
             "mkdir","rmdir", "rename", "getattr", "setattr",
             "getxattr", "setxattr", "statfs", "sync",
             "samedir_rename", "crossdir_rename", "punch"]
```




Since we keep the timestamp, we can also made training samples as time series, and train more complicated models such as recurrent neural networks in the future.

# Isolation Forest (1/2)

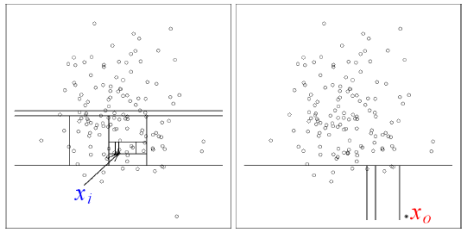


- Isolation Forest builds a set of proper binary trees from dataset: X = {x1, ..., xn} of n instances.
  - to build one isolation tree, recursively divide X by randomly selecting an attribute q and a split value p, until either: (i) the tree reaches a height limit, (ii) |X| = 1 or (iii) all data in X have the same values.
  - For a new data point, its path length h(x) of a iTree is the the number of edges x traverses an iTree from the root node until the traversal is terminated at an external node.
  - Than its anomaly score is

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}}$$

$$c(n) = 2H(n-1) - (2(n-1)/n)$$

$$H(i) = \ln(i) + 0.5772156649$$



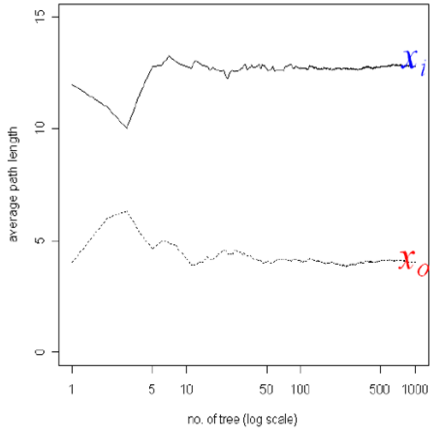
(a) Isolating $x_i$ (b) Isolating $x_o$

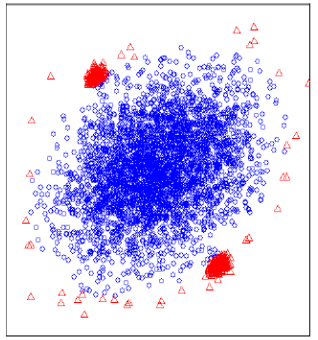
# Isolation Forest (2/2)



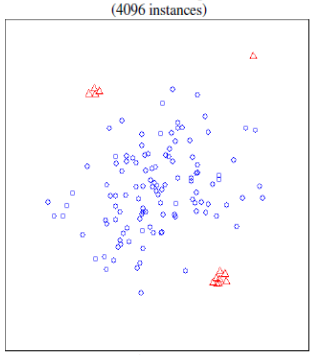
- Two parameters of Isolation Forest
  - number of trees ($t$)
  - number of samples used to build a Forest ($\psi$)
- Experiments show that
  - h(x) converges with a small number of trees
  - Training with sub sampling gives comparable result with training on full dataset
  - Default setting of $\psi$ is *256*
- very effective on large, high dimensional data
- Widely used, recommended by scikit-learn package

training stage: $O(t\psi \log \psi)$
evaluating stage $O(nt \log \psi)$



(a) Original sample
(4096 instances)



(b) Sub-sample
(128 instances)

# Implementation details



- Hyper parameters of IF
  - n_estimators=100, max_samples=256, contamination=0.1
- Separate models for Metadata and data operations
- Separate models with samples collected  last day and week
- New job samples are tagged by cron job every 10 mins
- Since we have detailed information of HT-Condor job in Elasticsearch DB, we can display details of abnormal jobs
  - uid, submission time, procname etc.,

# Visualization

- Tagged data samples for last 10 mins are reshuffled for visualization with matplotlib
- The first 5000 samples are reduced to 2-D with PCA and t-SNE

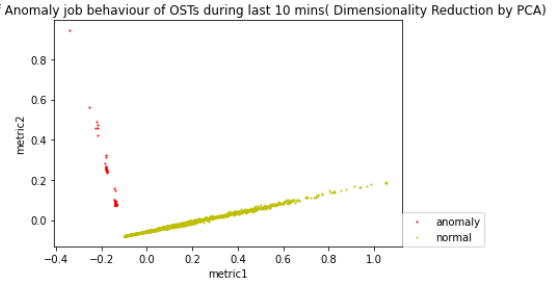
```
In [61]: adf_sorted
Out[61]:
```

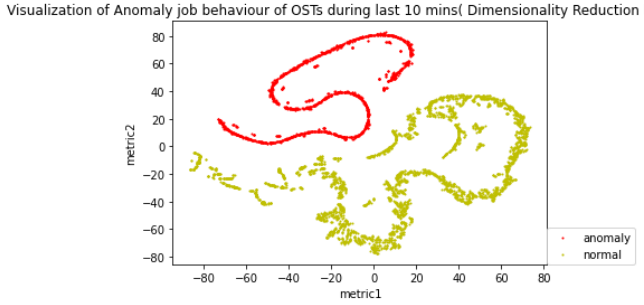
| | Unnamed: 0 | trigger | fsname | read_samples | write_samples | destroy | create | get_info | set_info | quotactl | sum_read_bytes | sum_write_bytes | sta |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 18070 | 3633 | 45626603_180 | bprofs | 0.000000 | 0.016658 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.000000 | 78.475183 | 16375 |
| 22816 | 3674 | 45626603_1504 | bprofs | 0.000000 | 0.016658 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.000000 | 68.363788 | 16375 |
| 23729 | 4587 | 45626605_593 | bprofs | 0.000000 | 0.016662 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.000000 | 68.595432 | 16375 |
| 20373 | 1231 | 45626603_971 | bprofs | 0.016656 | 0.016656 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 68.224225 | 68.257538 | 16375 |
| 4136 | 4136 | 45626604_27 | bprofs | 0.000000 | 0.016650 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.000000 | 67.798340 | 16375 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 35704 | 356 | 45626604_937 | bprofs | 0.000000 | 0.016664 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.000000 | 0.000000 | 16375 |
| 35481 | 133 | 45626603_1479 | bprofs | 0.000000 | 0.016664 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.000000 | 0.000000 | 16375 |
| 35533 | 185 | 45626603_613 | bprofs | 0.000000 | 0.016666 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.000000 | 0.000000 | 16375 |
| 23911 | 4769 | 45626605_694 | bprofs | 0.000000 | 0.000000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.000000 | 1337.916667 | 16375 |
| 4829 | 4829 | 45626605_667 | bprofs | 0.000000 | 0.000000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.000000 | 1267.866667 | 16375 |

1574 rows × 18 columns

```
In [62]: for i in range(0,10):
             jobid=adf_sorted.iloc[i,1].replace("_",".")
             print(jobid)
             job_info_json=get_jobinfo_json(jobid)
             res=search_job("condorpro",job_info_json)
             if (res!=[]):
                 print("%s,%s,%s,%s\n"%(jobid,res["fields"]["execnode"],res["fields"]["user"],res["fields"]["command"]))
```

```
45626603.180
job info not ready
45626603.1504
job info not ready
45626605.593
45626605.593,['bws0641.ihep.ac.cn'],['offline'],['boss.exe']

45626603.971
45626603.971,['bws0796.ihep.ac.cn'],['offline'],['boss.exe']

45626604.27
45626604.27,['bws0854.ihep.ac.cn'],['offline'],['boss.exe']

45626603.1284
45626603.1284,['bws0825.ihep.ac.cn'],['offline'],['boss.exe']

45626603.1134
job info not ready
45626603.620
45626603.620,['bws0887.ihep.ac.cn'],['offline'],['boss.exe']
```

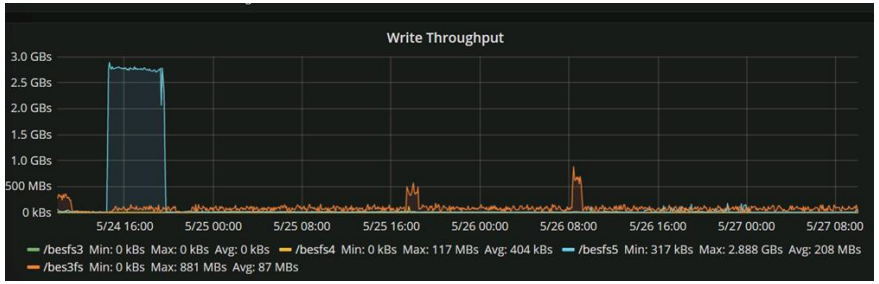


# Example of workload back trace



- In the right graph there is a peak write on the BESIII home directory
- If we sort jobs during that time span by its abnormal score predicted by last day model,
- Results shown that these abnormal jobs largely overlapped with a batch of jobs submitted by a same user and they have large write_bytes/sec




```
ostdayadf.iloc[:10,[0,1,2,3,4,5,6,7,8,9,10,13,14]]
```

| | trigger | fsname | read_samples | write_samples | destroy | create | get_info | set_info | quotactl | sum_read_bytes | sum_write_bytes | lastday_score |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 27003466_1255 | besfs5 | 1045.216667 | 581.616667 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 4.281207e+06 | 1.770720e+09 | 0.874030 |
| 27003466_1051 | besfs5 | 1065.733334 | 603.070160 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 4.365244e+06 | 1.840973e+09 | 0.874030 |
| 27003466_139 | besfs5 | 520.402149 | 562.585470 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 4.240043e+06 | 8.778144e+08 | 0.874030 |
| 27003466_825 | besfs5 | 419.933333 | 611.783333 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.428762e+06 | 5.720865e+08 | 0.872258 |
| 27003466_1651 | besfs5 | 464.408741 | 526.808741 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.781154e+06 | 6.993245e+08 | 0.872258 |
| 27003466_19 | besfs5 | 482.501990 | 1064.066666 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.928747e+06 | 7.546838e+08 | 0.872258 |
| 27003466_895 | besfs5 | 398.400000 | 1346.600000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.250517e+06 | 1.022176e+09 | 0.872258 |
| 27003466_1698 | besfs5 | 436.483333 | 472.370999 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.575671e+06 | 1.235001e+09 | 0.872258 |
| 27003466_1305 | besfs5 | 352.718972 | 403.518972 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.871842e+06 | 7.974343e+08 | 0.872258 |
| 27003466_551 | besfs5 | 378.716667 | 414.933333 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 3.114325e+06 | 4.720840e+08 | 0.872258 |

| | execnode | jobsubstrid | user | jobidsubtime | group |
|---|---|---|---|---|---|
| bws0732.ihep.ac.cn | 27040249.2 | | 2021-05-24 10:16:18 | physics |
| bws0881.ihep.ac.cn | 27003466.1051 | | 2021-05-23 22:27:59 | physics |
| bws0866.ihep.ac.cn | 27003466.1255 | | 2021-05-23 22:27:59 | physics |
| lhws169.ihep.ac.cn | 27003466.1230 | | 2021-05-23 22:27:59 | physics |
| bws0910.ihep.ac.cn | 27040249.81 | | 2021-05-24 10:16:18 | physics |
| ... | ... | | ... | ... |
| bws0921.ihep.ac.cn | 27003466.468 | | 2021-05-23 22:27:59 | physics |
| bws0910.ihep.ac.cn | 27003466.1050 | | 2021-05-23 22:27:59 | physics |
| hxm1010.ihep.ac.cn | 27003466.1833 | | 2021-05-23 22:27:59 | physics |
| jnws033.ihep.ac.cn | 27003466.1597 | | 2021-05-23 22:27:59 | physics |
| bws0835.ihep.ac.cn | 27040249.33 | | 2021-05-24 10:16:18 | physics |

# Outline

# Summary and next steps



- We implemented an automatic detection system of problematic I/O behaviors in IHEP cluster
- By data driven, unsupervised  machine learning, it can make detection with very little prior knowledge of the anomalies
- In the future, this system can be extended to use more compressive prediction models and wider applications in IT maintenance
  - Better user interface,  a human tagging entrance and implementation in a web framework other than Jupyter notebook

Thank You!
Questions?