CMSSpark for CMS metadata

Valentin Kuznetsov, Cornell Univ

CMS Computing & Offline meeting, April 2017

CMS metadata on HDFS

- CMS has the following metadata on HDFS:
  - DBS (*CVS*): /project/awg/cms/CMS_DBS3_PROD_{GLOBAL,PHYS0[1-3]}
  - PhEDEx (*CSV*): /project/awg/cms/phedex/block-replicas-snapshots
  - PhEDEx (*CSV*): /project/awg/cms/phedex/catalog
  - AAA (*JSON*): /project/monitoring/archive/xrootd/raw/gled
  - EOS (*JSON*): /project/monitoring/archive/eos/logs/reports/cms
  - CMSSW (*Avro*): /project/awg/cms/cmssw-popularity
  - JobMonitoring (*Avro*): /project/awg/cms/{jm-data-popularity,job-monitoring}
  - WMArchive (*Avro*): /cms/wmarchive/avro
- What we can do with them?

Use cases

- Site-utilization, e.g. what occupy my T[1-3]-site?
  - what is most popular release, which sites hold old/new/specific release(s), what are data-tier allocation on my site
  - can we get full statistics of all CMS T1+T2+T3 sites?
- Users activity/daily stats:
  - which data-tiers/datasets are accessed most by CMSSW/AAA/EOS
  - job throughput, failure distributions
- Derivatives of above, e.g. distribution of data-tiers at T2 sites

How to deal with data on HDFS

- VM with Hadoop eco-system setup
  - Java, Hadoop libraries, special rules for ip-tables, etc.
- Use CERN analytix cluster
- Write Python code, but **we lack of expertise and experience**
  - writing Python for data on HDFS is not trivial or at least is not what you get use to
  - you can easily lost in eco-system: Hive, Pig, Spark, Kafka, HBase, Sqoop, Flume, Impala
  - CERN IT provides some training, e.g. https://indico.cern.ch/event/590439

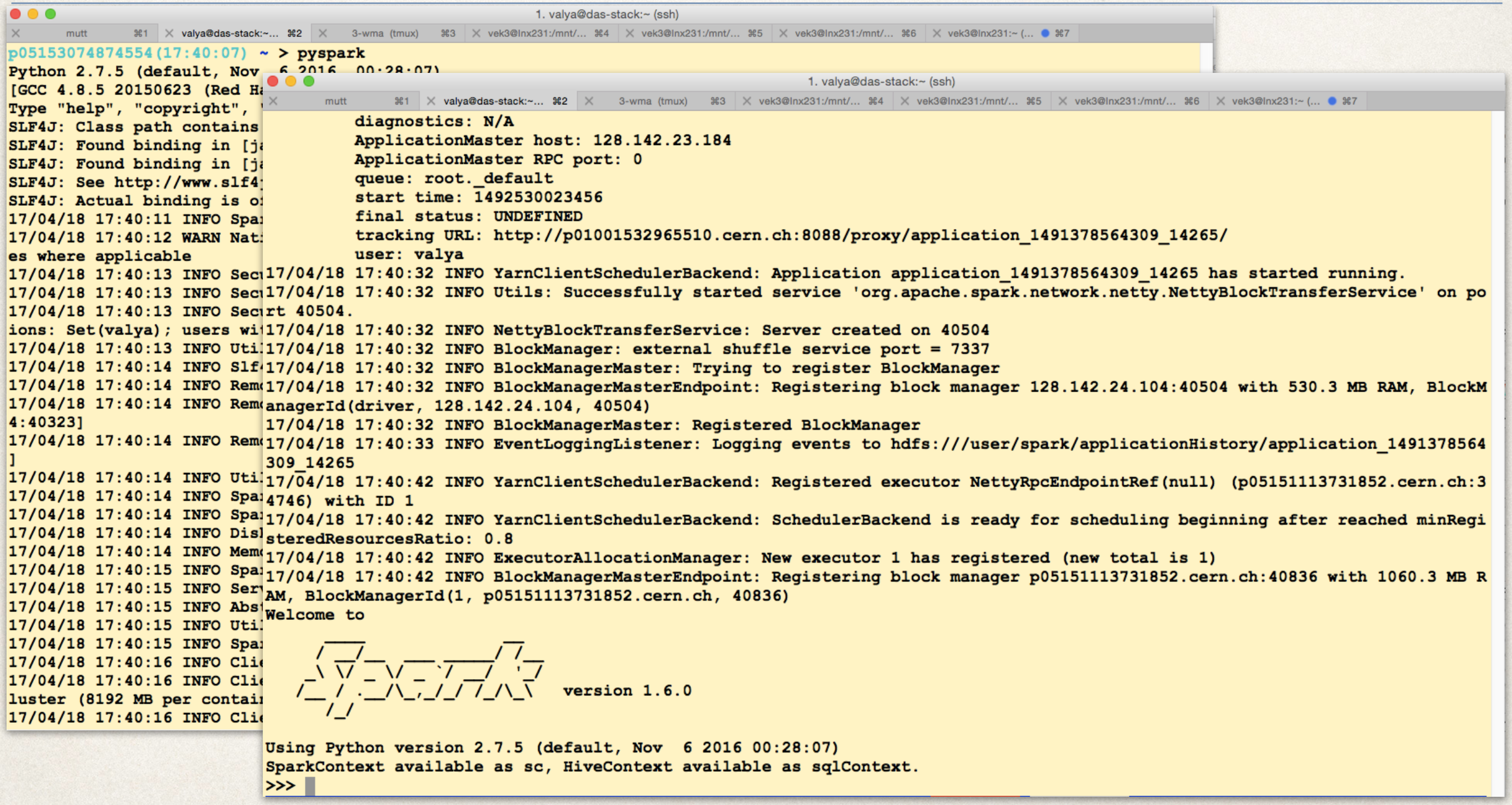
CERN IT analytix cluster

- CERN IT provides analytix cluster for experiment needs
  - 39 nodes with 64GB of RAM and 32 cores/node
    - Intel(R) Xeon(R) CPU E5-2650 0 @ 2.00GHz
    - AMD Opteron(TM) Processor 6276
    - Mix of CentOS 7 (27 nodes), SLC6 (12 nodes)
  - Request your analytix account via CERN SNOW ticket: https://cern.service-now.com/service-portal/service-element.do?name=Hadoop-Service
- CMS has its quota on cluster and is welcome to utilize its resources (ATLAS is doing this for a long time)

Before you start

- HDFS is not your local filesystem
  - if you run on Spark you may read from local disk but you'll write to HDFS and not on local disk (***your job runs on cluster nodes and not locally***)
- You interact with HDFS/Spark via PySpark
  - Python on Spark (PySpark) is a **wrapper** around Spark Java libs and will communicate with them within your shell or python code
  - your code should be complaint with python version of a worker node (***python 2.6.6/SLC6 and python 2.7.5/CentOS***)
  - 3rd party python libraries should be uploaded with your code to worker node
- So far we can't use custom python build and we don't have CVMFS on analytix cluster (***PySpark +Spark+Hadoop vs custom build vs experiement specifics vs OS***)
- You can use PySpark shell, but should submit your python program as a "job" via spark-submit/ pyspark wrappers, a la batch submission

Welcome to PySpark



PySpark basics

- **Spark aims to parallelize user jobs, run 100x faster than Hadoop MR in memory**
- Spark is written in Java and provides APIs for Java, Scala, Python and R languages
- PySpark is a **wrapper** around Spark libraries
- PySpark provides set of APIs to write code to access data on HDFS and use Spark platform
  - RDD: Resilient Distributed Dataset (low-level)
  - DataFrame: a data collection similar to pandas/R data frames (high-level)
- Writing code is not trivial exercise: you don't use loop to iterate over your data, and rather use Spark APIs to apply functions/actions to your (distributed) dataset, e.g.

```
def myfunc(row): yield row # special care should be done for 3d party libs

df.foreach(myfunc) # myfunc will be applied to each Row of DataFrame (df)
```

PySpark example, reading AAA

```
from pyspark import SparkContext
from pyspark.sql import Row, HiveContext, DataFrame

# setup Spark Context with distinguished name & create sql context, a la database
ctx = SparkContext(appName='cms')
sqlContext = HiveContext(ctx)
files = ['hdfs:///cms/path/file1', 'hdfs:///cms/path/file2', …]

# Read data from path on HDFS and convert them into RDD
rdd = reduce(DataFrame.unionAll, [sqlContext.jsonFile(path) for path in files])

# perform data transformation depending on data structure stored on HDFS
aaa_rdd = rdd.map(lambda r: r['data'])

# convert RDD into a DataFrame and register it as a table
aaa_df = sqlContext.createDataFrame(aaa_rdd)
aaa_df.registerTempTable('aaa_df')

# write SQL query
stmt = "SELECT * from aaa_df"
df = sqlContext.sql(stmt)
df.write.format("com.databricks.spark.csv").option("header", "true").save(fout)
```

PySpark execution

```
# set Java libraries to read your data
csvjar=/afs/cern.ch/user/l/lmeniche/public/spark-csv-assembly-1.4.0.jar
avrojar=/usr/lib/avro/avro-mapred.jar
sparkexjar=`ls /usr/lib/spark/examples/lib/spark-examples* | tail -1`

# You need wrapper around python since nodes are mix of python2.6 & 2.7
PYSPARK_PYTHON='/afs/cern.ch/user/v/valya/public/python27' \
spark-submit --jars $csvjar,$avrojar,$sparkexjar \
        --master yarn-client \
        --driver-class-path '/usr/lib/hive/lib/*' \
        --driver-java-options '-Dspark.executor.extraClassPath=/usr/lib/hive/lib/*' \
        --executor-memory 5g \
        <path>/your_script.py ${1+”$@"}
```

And you’re ready to go!!!

But I bet your code may fail frequently and you may spend endless hours (as I did) to debug python+spark+Java issues

PySpark caveats

- Careful crafting of data is required to fit your data into worker node memory
  - most of errors you'll experience are java.lang.OutOfMemoryError, Java heap, GC errors, e.g. never collect intermediate results, instead perform series of operations over data and either write them out to HDFS or aggregate.
  - the same job may run or fail depending on resource utilization and amount of data and operations you're doing to process your data
- To take advantage of parallelism you should not iterate over containers, instead you should operate over them, e.g. `df.foreach` instead of `for item in df`
- No global scope within container operations
- Normal python function may not work well (be very slow) within DataFrames operations, instead you must use pyspark equivalents, e.g. `split` vs `pyspark.sql.functions.split`
- PySpark 1.X != PySpark 2.X, read proper docs, CERN IT cluster is PySpark 1.6
- Lack of python support for new Kafka streams (affect streaming within Spark job)

Is it worth it?

github.com/vkuznet/CMSSpark

- Aim to help with complexity of PySpark and its submission infrastructure
  - Set of utilities for HDFS I/O
  - Common schema definitions for CMS HDFS metadata streams
  - PySpark submission script & sending data to CERN MONIT system
- Examples:
  - reading CSV, Avro, JSON data from HDFS
  - writing data back to HDFS or CERN MONIT
  - data-conversion, RDD, DataFrames, SQL tables
  - merging metadata streams via SQL queries, e.g. DBS+PhEDEx, DBS+CMSSW, etc.

Logistics

```
ssh analytix

git clone git@github.com:vkuznet/CMSSpark.git; cd CMSSpark

export PYTHONPATH=$PYTHONPATH:$PWD/src/python; export PATH=$PWD/bin:$PATH

run_spark dbs_phedex.py -fout=hdfs:///cms/users/NAME/dbs_phedex -yarn

run_spark dbs_phedex.py -fout=hdfs:///cms/users/NAME/dbs1_phedex -yarn -inst=phys01

run_spark dbs_cmssw.py -fout=hdfs:///cms/users/NAME/cmssw -date=20170411 -yarn

run_spark dbs_aaa.py -fout=hdfs:///cms/users/NAME/aaa -date=20170411 -yarn

run_spark dbs_eos.py -fout=hdfs:///cms/users/NAME/eos -date=20170411 # run on local node

run_spark dbs_jm.py -fout=hdfs:///cms/users/NAME/jm -date=20170411 -yarn # run on cluster
```

Daily stats in ~10 min

CMSSW: combine CMSSW job stats with DBS and PhEDEx info

```
SITE_NAME,count,date,count_type,primds,procds,tier
T2_DE_DESY,954,1.4918616E9,cmssw,JetHT,Run2016F-03Feb2017-v1,MINIAOD
T3_US_Colorado,10,1.4918616E9,cmssw,HLTPhysics0,Run2016H-v1,RAW
```

EOS: combine EOS file stats with DBS info

```
count,date,count_type,primds,procds,tier
16,1.4918616E9,eos,HIMinimumBias1,HIRun2015-v1,RAW
4,1.4918616E9,eos,Tau,Run2016D-03Feb2017-v1,MINIAOD
```

AAA: combine AAA file stats with DBS info

```
count,date,count_type,primds,procds,tier
659,1.4918616E9,aaa,MuOniaParked,Run2012C-22Jan2013-v1,AOD
71,1.4918616E9,aaa,HIAllPhysics,HIRun2010-v1,RAW
```

Results show data for 20170411 and aggregation with DBS global

JM: combine JobMonitoring file stats with DBS info

```
SiteName,JobExecExitCode,FileType,Type,tot_cpu,ecode_count,tot_wc,file_type_count,type_count,date,count_type,primds,procds,tier
T1_US_FNAL,0,EDM,analysis,328022.22,114,968717.0,114,114,1.4918616E9,jm,ZeroBiasBunchTrains4,Run2016H-v1,RAW
T2_CH_CERN,0,EDM,analysis,25039.009999999995,105,33302.0,105,105,1.4918616E9,jm,Tau,Run2016B-03Feb2017_ver2-v2,MINIAOD
```

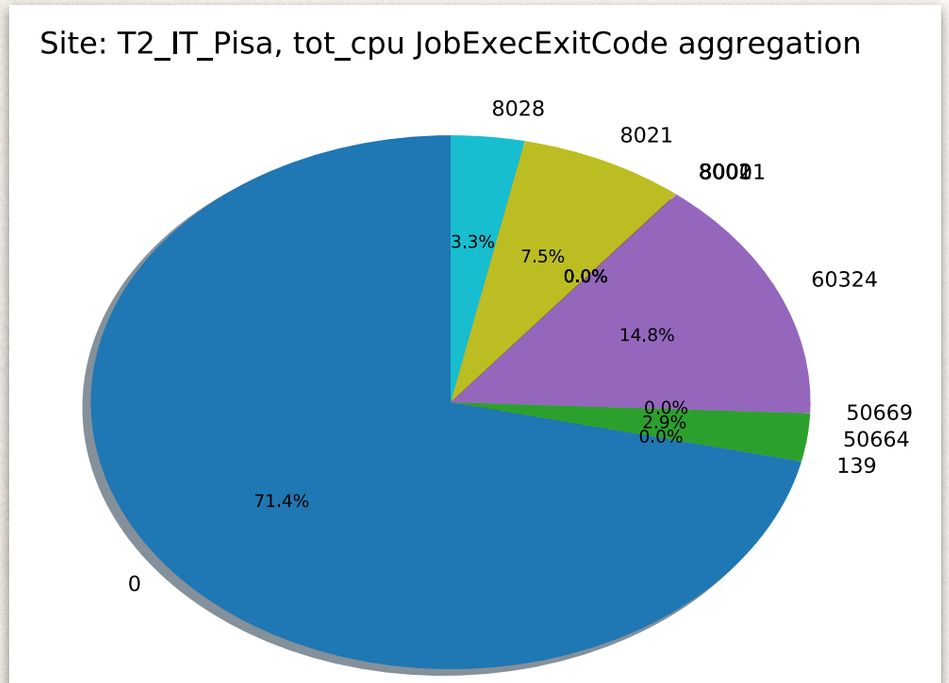
Data access daily stats

```
import pandas as pd

# read data
eos = pd.read_csv(“eos.csv”)
aaa = pd.read_csv(“aaa.csv”)
cmssw = pd.read_csv(“cmssw.csv”)

# drop SITE_NAME in order to concatenate DF’s
cmssw = cmssw.drop(“SITE_NAME”, axis=1)

# concatenate DF’s (they have now similar structure)
df = pd.concat([aaa,eos,cmssw])

# group by data-tier
gb = df.groupby(‘tier’)

# aggregate all counters
data = gb[‘count’].agg(np.sum)

# print final data frame
print(data.sort_values(ascending=False))
```

```
tier
AOD                  1196449
MINIAOD               256448
MINIAODSIM            185249
AODSIM                115635
GEN-SIM-RECO           69181
RAW                    57490
GEN-SIM-RAW            19341
RAW-RECO               10109
GEN-SIM                 8884
ALCARECO                1885
GEN-SIM-DIGI-RAW        1276
USER                     707
LHE                      247
RAWAODSIM                117
FEVT                      59
GEN-SIM-DIGI-RECO         22
RECO                       4
DQM                        3
Name: count, dtype: int64
```

91%

Job Summary daily stats



See jm_stats.py for code details

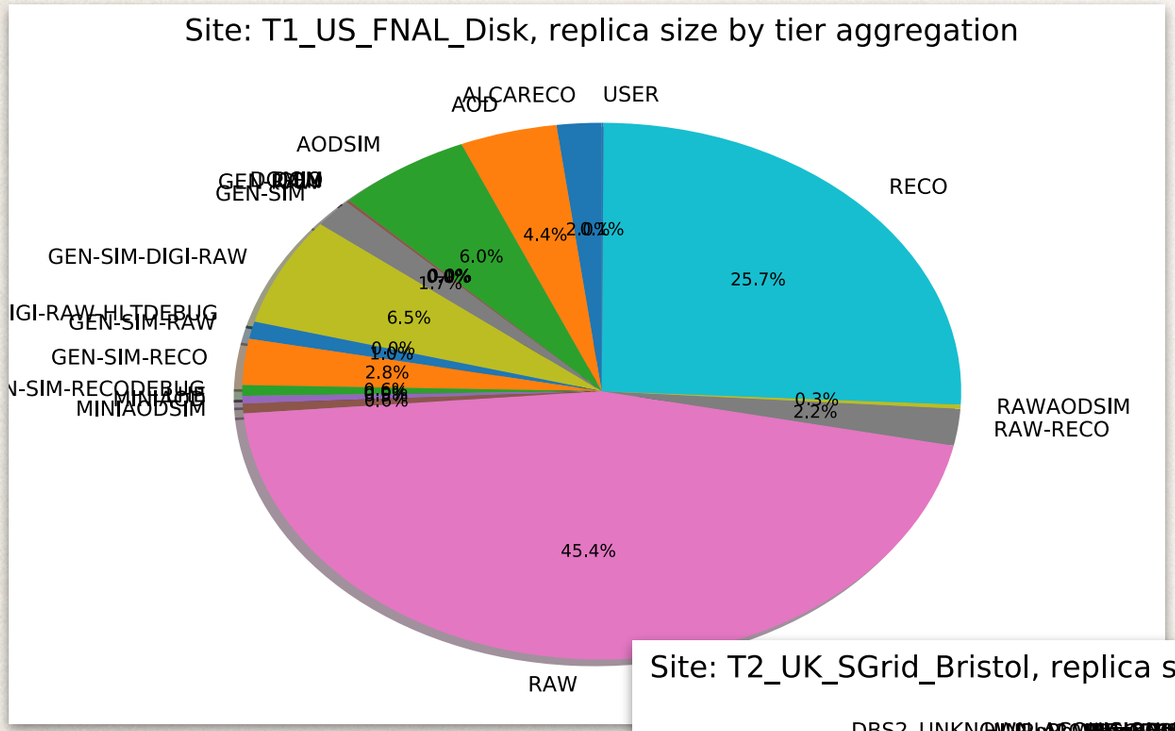
T1_ES_PIC

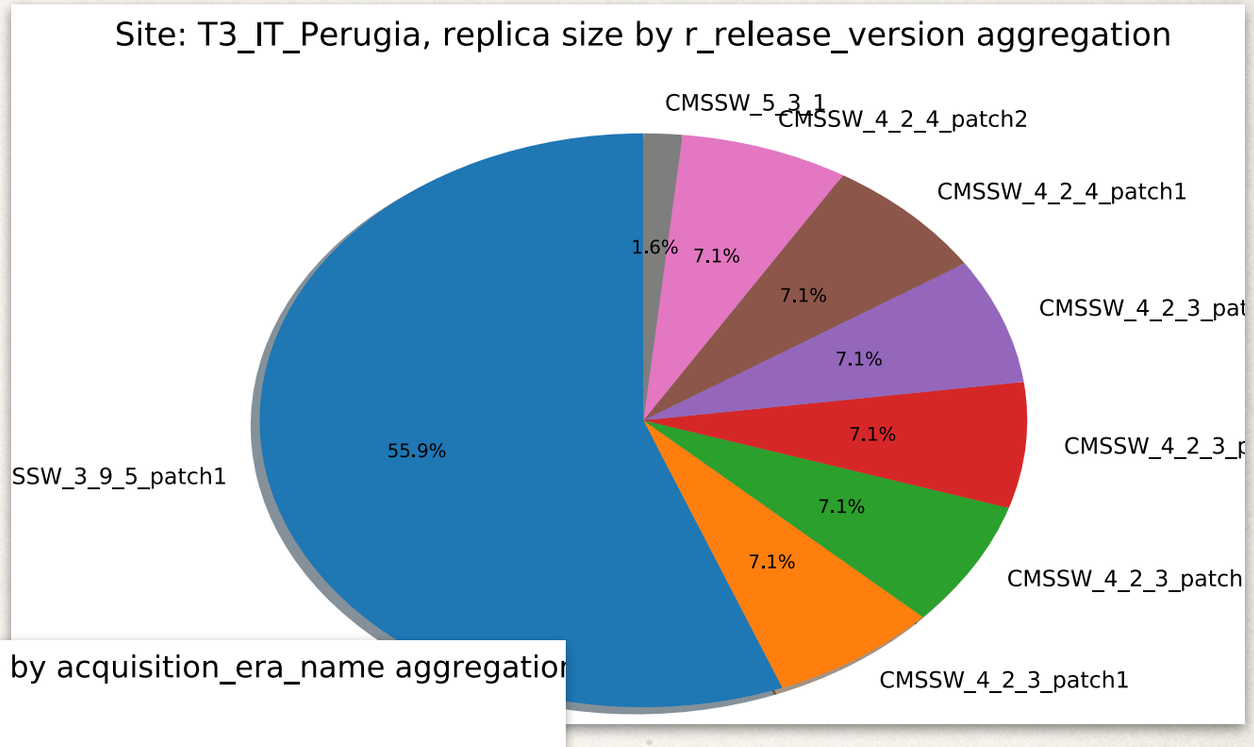
JobExecExitCode	tot_cpu	ecode_count	tot_wc
0	2607015.58	1332	5737309.0
50664	20707.71	20	1625222.0
8001	16148.62	52	357181.0
8021	23657.17	20	834893.0
8028	48637.65	42	2606294.0

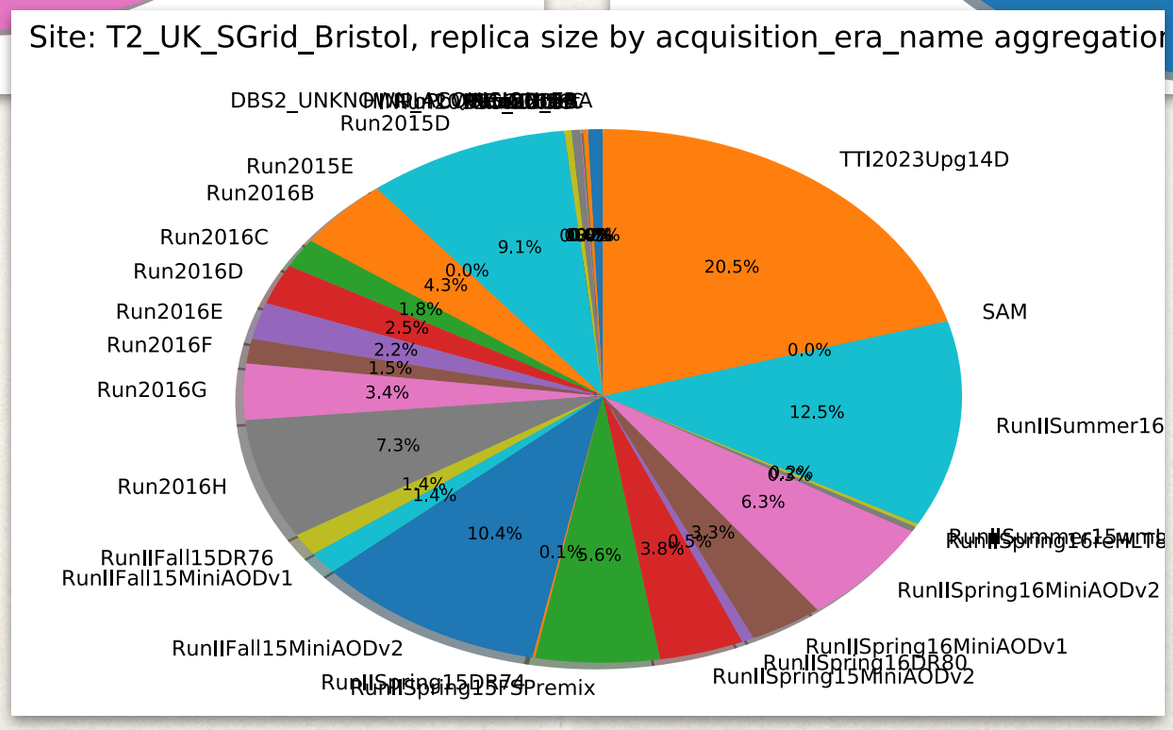
T1_FR_CCIN2P3

JobExecExitCode	tot_cpu	ecode_count	tot_wc
0	39013280.31	5914	48552294.0
134	8236.20	3	132690.0
60321	1392658.46	40	1411063.0
60324	14749234.97	1906	15681556.0
8001	7539.90	17	47287.0
8021	462015.99	63	1604423.0
8028	11254.20	1	11608.0

Site utilization by tier, release, era







Get stats from DBS+PhEDEx for all CMS sites in 10-20 minutes

See stats.py for code details

Site utilization, tabular form

Results from DBS global instance sorted by PhEDEx Replica size

T2-sites, evts=2508093123707, dataset size=1.283e+17 (113.989 PB)
replica size=7.446e+16 (66.137 PB)

tier	evts	size	pbr_size
AOD	162782404863	2.031304e+16	1.959890e+16
RAW	1683921651218	4.498240e+16	1.722459e+16
AODSIM	34936712479	9.451795e+15	9.058594e+15
RECO	55951414059	1.714168e+16	7.169092e+15
MINIAOD	246201671780	4.792439e+15	4.780326e+15
MINIAODSIM	132260740917	4.145671e+15	4.075050e+15
GEN-SIM	35102730685	1.502151e+16	3.213296e+15
ALCARECO	95433911221	3.302962e+15	3.095442e+15
GEN-SIM-RAW	1325378909	1.730448e+15	1.601045e+15
GEN-SIM-RECO	867610238	1.215509e+15	9.514087e+14
RAW-RECO	687922996	1.056303e+15	7.773873e+14
GEN-SIM-DIGI-RAW	951986805	1.856123e+15	7.366950e+14
USER	2444128149	7.315808e+14	7.169075e+14
GEN-SIM-RECODEBUG	419892501	6.901112e+14	6.714903e+14
RAWAODSIM	370090370	4.393273e+14	4.319676e+14
FEVT	1195402471	1.091112e+15	1.569082e+14
GEN	8455409529	1.805733e+14	1.042719e+14
LHE	36371033202	1.290807e+14	4.294228e+13
ALCAPROMPT	7350814504	2.571447e+13	2.621732e+13
GEN-SIM-RAW-RECO	4470042	2.204419e+13	2.204419e+13
FEVTHLTALL	13949607	1.491425e+13	4.245913e+12
GEN-SIM-DIGI-RAW-HLTDEBUG	913697	2.832780e+12	2.808071e+12
GEN-SIM-DIGI-RECO	635000	2.296991e+12	1.864344e+12
DQMIO	0	7.262767e+11	6.023728e+11
DQM	1042248465	6.835633e+10	6.475834e+10

T2_CN_Beijing, evts=4385981737,
dataset size=8.504e+14 (0.755 PB),
replica size=4.776e+14 (0.424 PB)

tier	evts	size	pbr_size
AOD	1837680818	3.073608e+14	2.860339e+14
AODSIM	291532057	9.067561e+13	7.922197e+13
RECO	262739619	1.922921e+14	3.211212e+13
MINIAODSIM	743700997	2.176906e+13	2.176299e+13
MINIAOD	799773539	2.085274e+13	2.085274e+13
GEN-SIM	263109521	1.879252e+14	1.165400e+13
GEN-SIM-RECO	2144385	1.396471e+13	1.056114e+13
ALCARECO	150800146	7.275727e+12	7.273371e+12
GEN-SIM-RAW	2851433	5.129433e+12	5.129433e+12
RAWAODSIM	1753081	2.467996e+12	2.467996e+12
GEN-SIM-RECODEBUG	79980	5.118645e+11	5.118645e+11
LHE	29816161	1.269364e+11	1.229156e+09
DQMIO	0	3.271108e+07	3.271108e+07

Stats files

- If you're interested in statistics for **ALL** CMS sites
- Site stats by data-tiers: http://bit.ly/2pxjDnY & http://bit.ly/2o0wuz0
- Site stats by acq. era: http://bit.ly/2pxzNxt & http://bit.ly/2o0CS9C
- Site stats by releases: http://bit.ly/2oSN7fi & http://bit.ly/2o0NLrR
- Site stats by jobs: http://bit.ly/2ocudAR & http://bit.ly/2oRX0Jy

Summary & future direction

- CMSSpark is ready for prime time but may require further tuning depending on use case(s)
- Available metadata on HDFS: DBS (global+phys0[1-3] instances), PhEDEx, CMSSW, AAA, EOS, JobMonitoring, WMArchive
- We can crunch largest databases (DBS+PhEDEx) in about half an hour and extract useful statistics
- In 10 minutes we can get full daily stats of CMSSW, AAA, EOS utilization & jobs throughput
- Validate data streams, identify useful metrics and start aggregate and stream data to CERN MONIT on a regular basis (work for summer student)
- Other sources can be added into common framework to accommodate new use cases
- Migrate under DMWM umbrella and support it officially

Back-up slides

Hadoop commands

```
# list content of the directory
hadoop fs -ls /project/awg/cms/CMS_DBS3_PROD_GLOBAL

# file operations
hadoop fs -cat /cms/users/vk/aaa_phys03/part-00199
hadoop fs -tail /cms/users/vk/aaa_phys03/part-00199
hadoop fs -get /cms/users/vk/aaa_phys03/part-00199 .
hadoop fs -put file.txt /cms/users/vk/

# check disk space usage (quota)
hadoop fs -count -q /cms/users/vk/aaa_phys03

# More HDFS commands at http://bit.ly/2pVWI2R

hadoop fs # deals with any FileSystem (HDFS, Local) while
hdfs dfs # for operations related to HDFS
```

How to get stats

```
# get headers first
hadoop fs -cat /cms/users/vk/datasets/part-00001 | head -1 > data.csv

# get data rows
hadoop fs -cat /cms/users/vk/datasets/part-* | grep -v primds |sort|uniq >> data.csv
```

```
# download/copy your data.csv into your favorite location

# feed data into R or python pandas, CMSSpark provides pandas script
git clone git@github.com:vkuznet/CMSSpark.git

# aggregate by data-tiers
CMSSpark/src/python/CMSSpark/stats.py --fin=data.csv –agg=tier

# aggregate by CMSSW releases
CMSSpark/src/python/CMSSpark/stats.py --fin=data.csv –agg=release

# aggregate by acquisition era
CMSSpark/src/python/CMSSpark/stats.py --fin=data.csv --agg=era
```