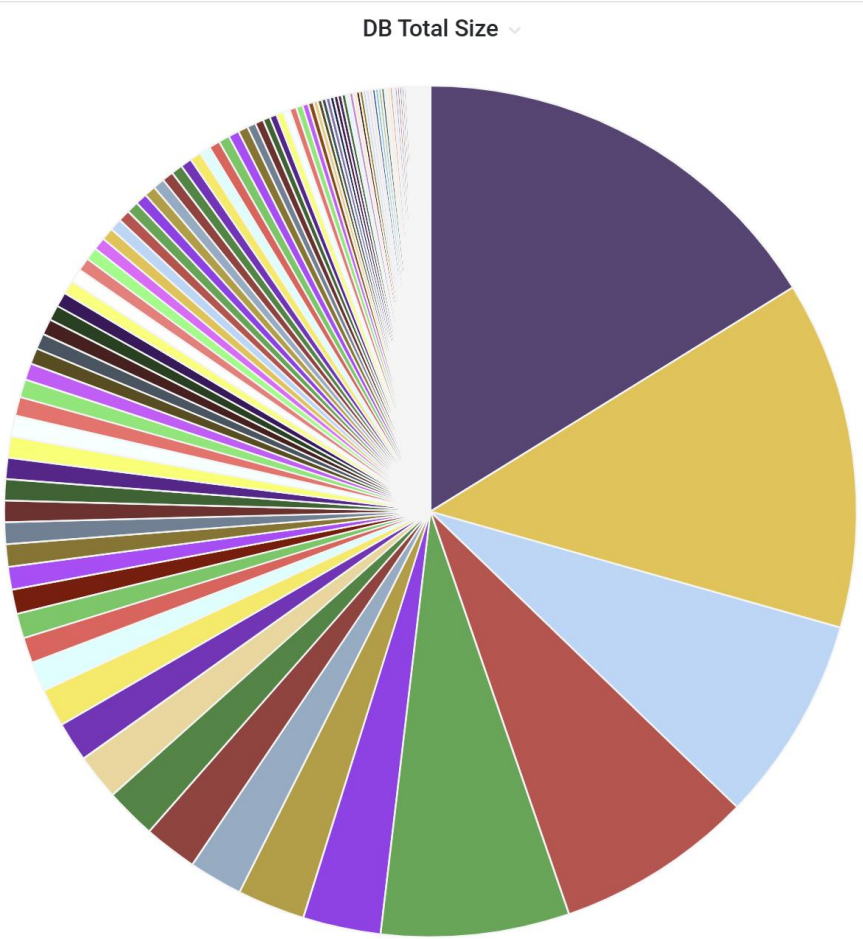
BNL Rucio DB Operation
by highly biased non-db admin

Hironori Ito
Brookhaven National Laboratory

RUCIO Workshop at UCSD, Sept 30 – Oct 4, 2024

@BrookhavenLab

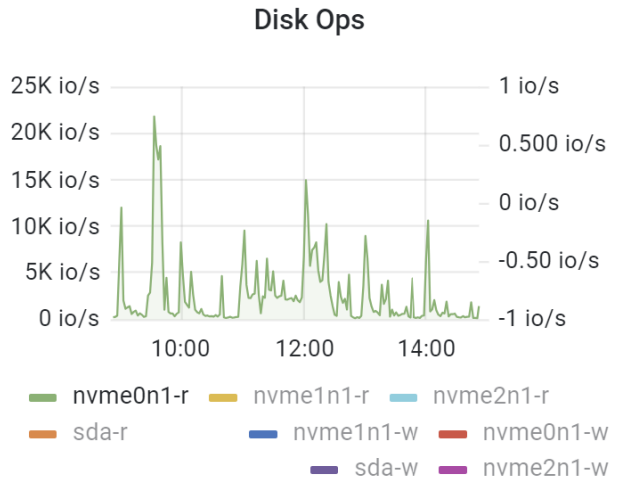
Rucio DB SIZE

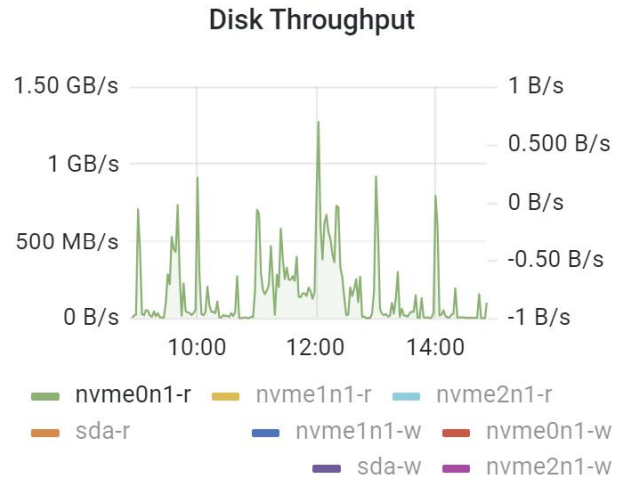


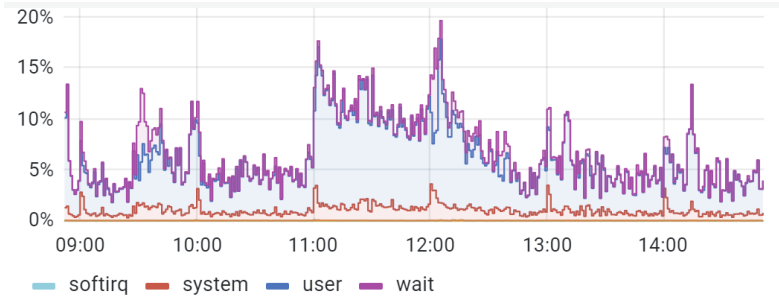
	ROWS	current	percentage
contents	106M	210 Bil	16%
replicas	109M	172 Bil	13%
locks	91M	102 Bil	8%
dids	105M	98 Bil	8%
did_meta	72M	92 Bil	7%
rules	14M	38 Bil	3%

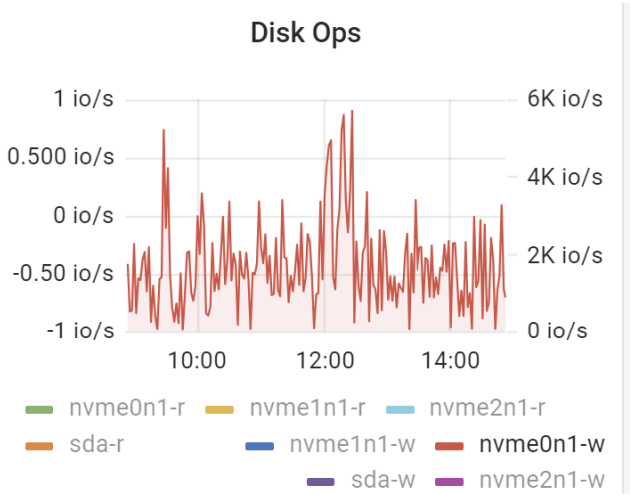
~100M files
1.5TB NVMe
196GB RAM
PostgreSQL 12
RHEL7

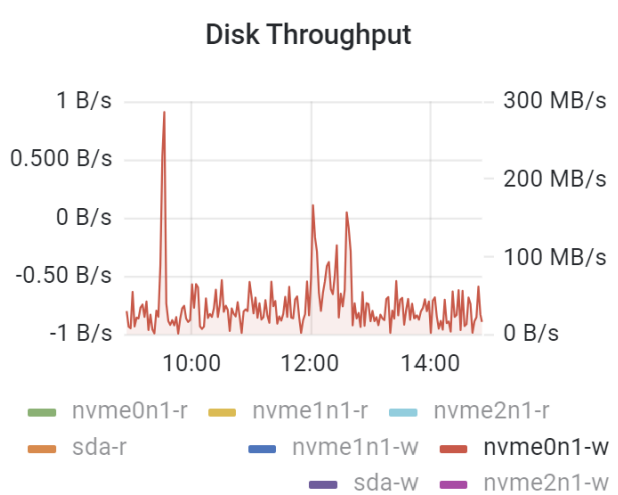
Observed Host Performance for RUCIO PostgreSQL







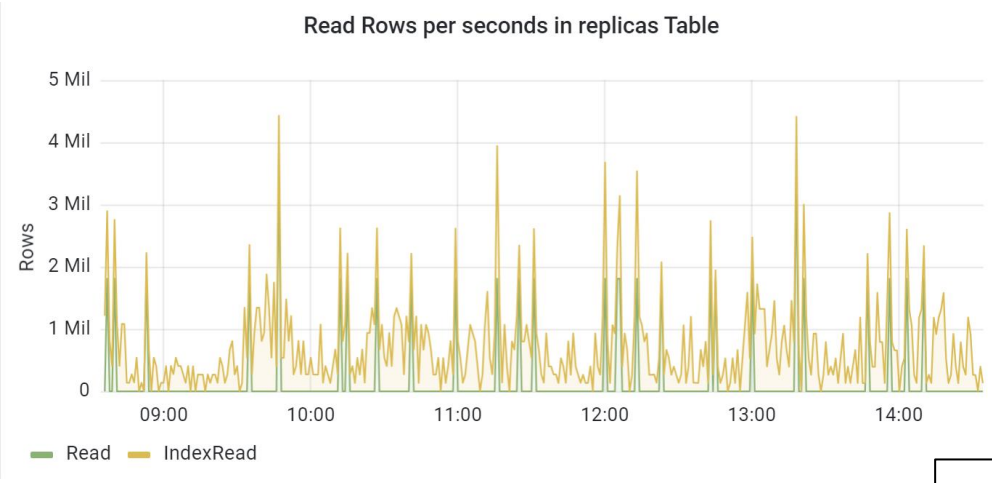


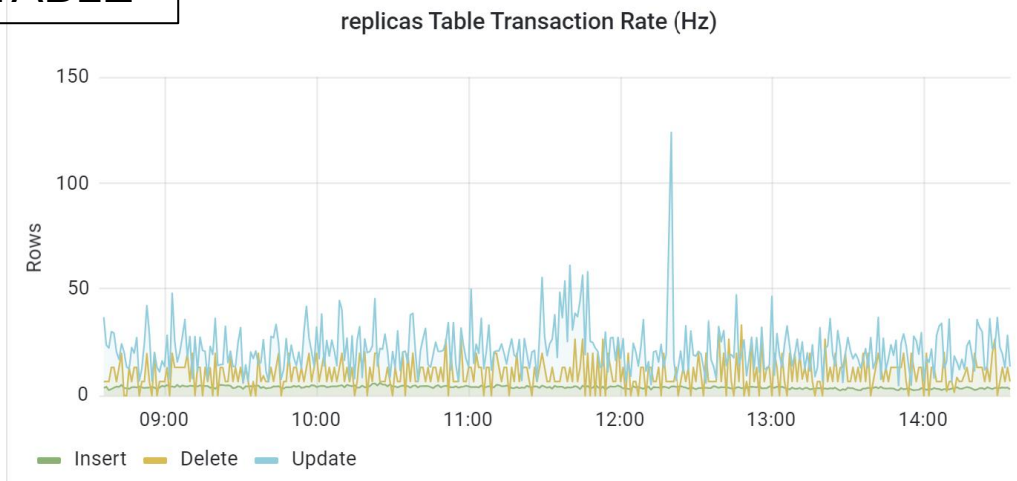


Quite Busy DB disks
~ 10s KIOPS
~ 100s MB/s

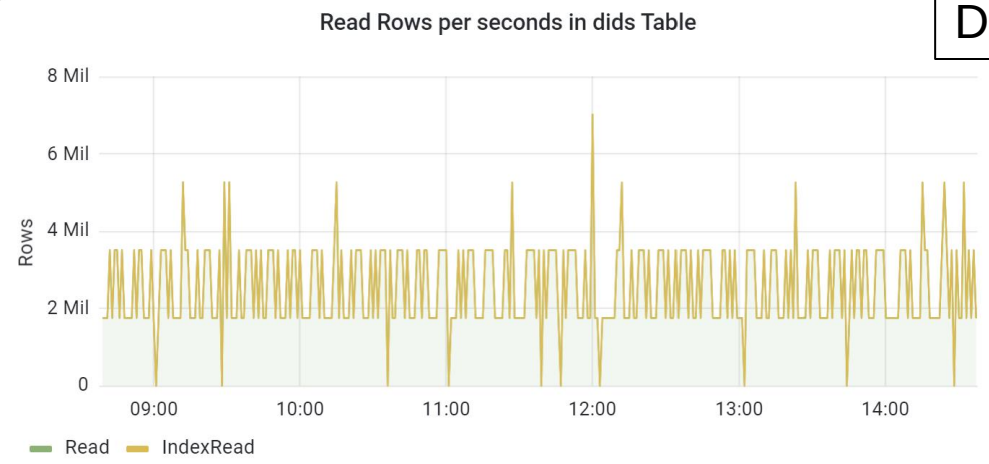
DB Transaction Rate

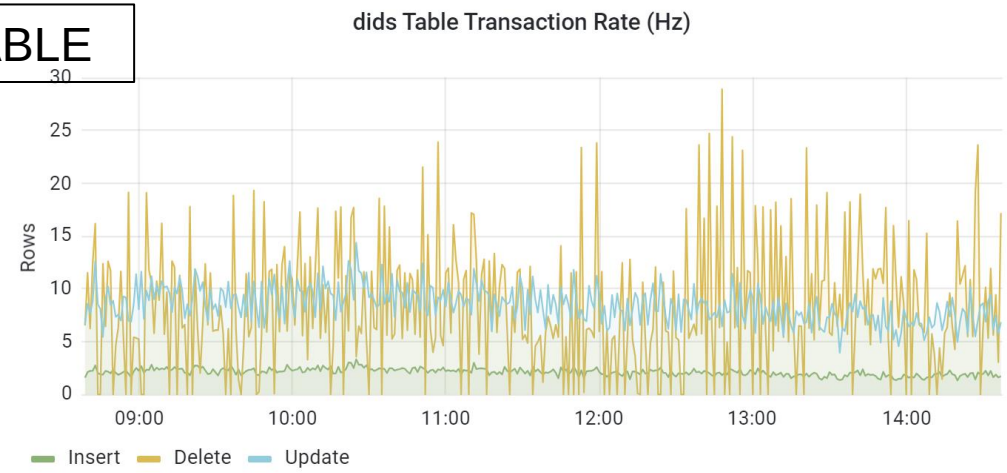
REPLCIA TABLE





DID TABLE



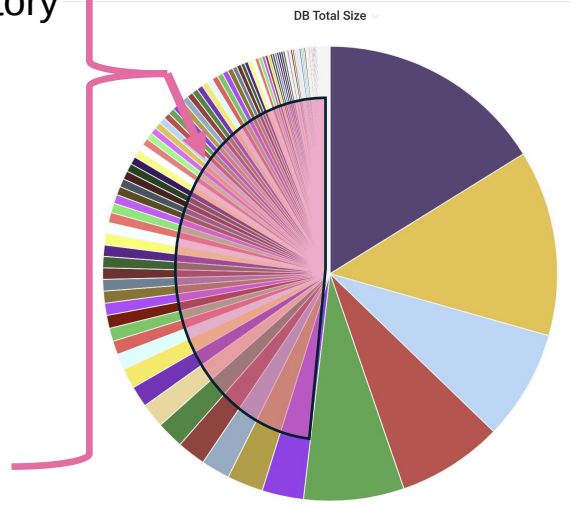


PostgreSQL WAL Segment Size

- Default WAL (write ahead log) segment size is 16MB
- It is too small for PostgreSQL replication if RUCIO is operating at high rate of change (like deletion) and if PostgreSQL is configured to do streaming backup.
  - The volume containing WAL segments grow quickly.
- WAL segment replication in PostgreSQL is done sequentially from the main system to the backup system.
- The process is too slow under high rate.
  - E.g. Deleting millions of user datasets and their files.
- The WAL segment size is adjustable
  - At the time of database creation
    - Initdb –wal-segisze=SIZE
  - Or, after stopping the database
    - pg_resetwal --wal-segsize=SIZE /var/lib/pgsql/…./data
    - Then, you need to restablish the replication (remake the backup. Painful!)
  - Cons: Higher chance that the backup is not up to seconds. But, does it matter if the backup is 30 seconds behind instead of 5 seconds? And, if the replication is not catching up, then, it is already behind.

RUCIO DB in PostgreSQL

- Lots of history tables
  - account_usage_history
  - archive_contents_history
  - configs_history
  - contents_history
  - messages_history
  - quarantined_replicas_history
  - replicas_history
  - requests_history
  - rse_usage_history
  - rules_history
  - rules_history_recent
  - sources_history
  - subscriptions_history



- All of these tables should be partitioned from the beginning by installation and codes!!!

- Some of these tables grow very quickly.
  - They will take over your database partition.
- Who will look at history from 3 years ago?
  - Most history are not used or useful by the operation
- Partitioned tables can be dropped instantly.
  - Instead of delete from tables where time < myrange
  - drop table my-partitioned table

Brookhaven National Laboratory

Partitioning history tables

```
CREATE TABLE rucio.rules_hist_recent (
    id uuid,
    scope character varying(25),
    name character varying(250),
    created_at timestamp without time zone,
    );
…;
CREATE INDEX "RULES_HIST_RECENT_ID_IDX" ON rucio.rules_hist_recent
USING btree (id);

CREATE INDEX "RULES_HIST_RECENT_SC_NA_IDX" ON
rucio.rules_hist_recent USING btree (scope, name);
```

created_at is not index → **Very Slow** “delete rules_hist_recent where created_at <‘XXXX-YY-DD’ “

```
CREATE TABLE rucio.rules_hist_recent (
…) partition by range(created_at);
….
create table RHR_2021_01 partition of rules_hist_recent for values from ('2021-01-01 00:00:00') to ('2021-02-01 00:00:00');

create table RHR_2021_02 partition of rules_hist_recent for values from ('2021-02-01 00:00:00') to ('2021-03-01 00:00:00’);

…
Instead of delete from rules_hist_recent where created_at < ‘2021-02-01’

Drop table RHR_2021_01
```

```
CREATE TABLE rucio.messages_history (
id uuid,
created_at timestamp without time zone,
…);


CREATE TABLE rucio.messages_history (
…) PARTITION BY RANGE (created_at);


CREATE TABLE rucio.msghist_2024_01 PARTITION OF
rucio.messages_history FOR VALUES FROM ('2024-01-01 00:00:00') to ('2024-02-01 00:00:00’);

….
Insteaed of delete from messages_history where created_at < ‘2024-02-01’

Drop table msghist_2024_01
```

Brookhaven National Laboratory

SQLAlchemy

SQLAlchemy

[PostgreSQL — SQLAlchemy 2.0 Documentation]

PostgreSQL Table Options

Several options for CREATE TABLE are supported directly by the PostgreSQL dialect in conjunction with the `Table` construct:

- `INHERITS`:

```
Table("some_table", metadata, ..., postgresql_inherits="some_supertable")

Table("some_table", metadata, ..., postgresql_inherits=("t1", "t2", ...))
```

- `ON COMMIT`:

```
Table("some_table", metadata, ..., postgresql_on_commit='PRESERVE ROWS')
```

- `PARTITION BY`:

```
Table("some_table", metadata, ...,
      postgresql_partition_by='LIST (part_column)')

.. versionadded:: 1.2.6
```

- `TABLESPACE`:

Brookhaven National Laboratory

Partitioning for lazy

- One need to pre-make the partition corresponding to the entry.
- How to automate
  - Just make one for the next year by scripts.
  - Run a script in cron
  - Add a trigger
  - Pg_partman https://github.com/pgpartman/pg_partman

But, maybe even better (aka lazier, maybe smarter) option.

TimescaleDB

From Wiki

TimescaleDB(https://www.timescale.com/) is an open-source time series database developed by Timescale Inc. It is written in C and extends **PostgreSQL**. TimescaleDB is a relational database and supports standard SQL queries. Additional SQL functions and table structures provide support for time series data oriented towards storage, performance, and analysis facilities for data-at-scale.

No need to learn a new query language

No ~~InfluxQL~~

No ~~PromQL~~

Aggregate functions over time!

Testing TimescaleDB

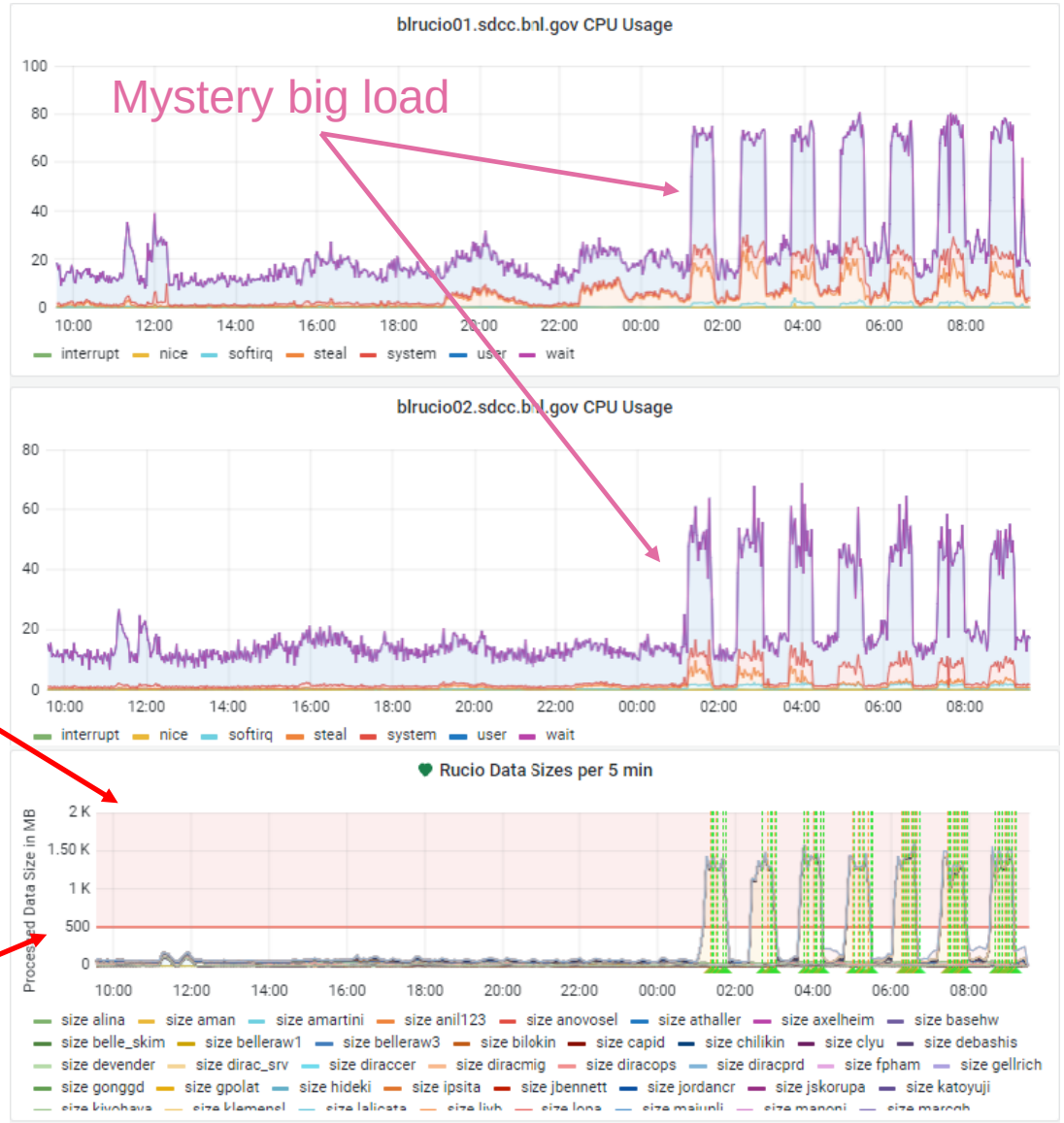
In Grafana monitor

```
SELECT  time_bucket('5 minutes', logtime) as time,
        sum(size)/1000000 as size,
        username
FROM
        ruciorequests
WHERE
        $__timeFilter(logtime)
GROUP by time, username
ORDER by time
```

Ruciorequests table by parsing Rucio HTTP server log

- logtime, clienthost, serverhost, httpmeth, httpapi, username, clientversion, clientmeth, httpcode, size, duration

Trigger Level
In Grafana



Partition Rucio table by TimescaleDB

- Regular partition
  - CREATE TABLE rucio.messages_history (…) **PARTITION BY RANGE (created_at);**
- TimescaleDB
  - CREATE TABLE rucio.messages_history (…)
  - SELECT **create_hypertable**('messages_history', by_range(created_at));

- It is partitioned by "time"
  - No need to create partition by hand or trigger
  - Default partition (aka chunk) is 7 days. Can be adjusted.
- Removing old data
  - If it was regular partition,
    - Drop table msghist_2024_01
  - TimescaleDB
    - Select drop_chunks('messages_history', older_than => '2024-02-01')

Brookhaven National Laboratory

SQLAlchemy TimescaleDB

[sqlalchemy-timescaledb · PyPI]

SQLAlchemy TimescaleDB

pypi package 0.4.1 | Tests passing | codecov 100% | downloads 165k

This is the TimescaleDB dialect driver for SQLAlchemy. Drivers `psycopg2` and `asyncpg` are supported.

Install

```
$ pip install sqlalchemy-timescaledb
```

Usage

Adding to table `timescaledb_hypertable` option allows you to configure the hypertable parameters:

Possible more use

- Replicas table

```
rse_id      | uuid                        |          | not null |
scope       | character varying(25)       |          | not null |
name        | character varying(250)      |          | not null |
bytes       | bigint                      |          |          |
md5         | character varying(32)       |          |          |
adler32     | character varying(8)        |          |          |
path        | character varying(1024)     |          |          |
state       | character varying(1)        |          |          |
lock_cnt    | integer                     |          | 0
accessed_at | timestamp without time zone |          |
tombstone   | timestamp without time zone |          |
created_at  | timestamp without time zone |          |          |
updated_at  | timestamp without time zone |          |
```

- If it were TimescaleDB

```
Select time_bucket('1 hour', updated_at) as bucket,
Sum(bytes), scope
From replicas
Where created_at > 'xxx' and created_at <='yyy' and rse_id='ABC' and state='ZYX'
Group by bucket, scope
Order by bucket
```

Directly plot on Grafana monitor

One can join with multiple tables
eg, rses, scopes, etc…

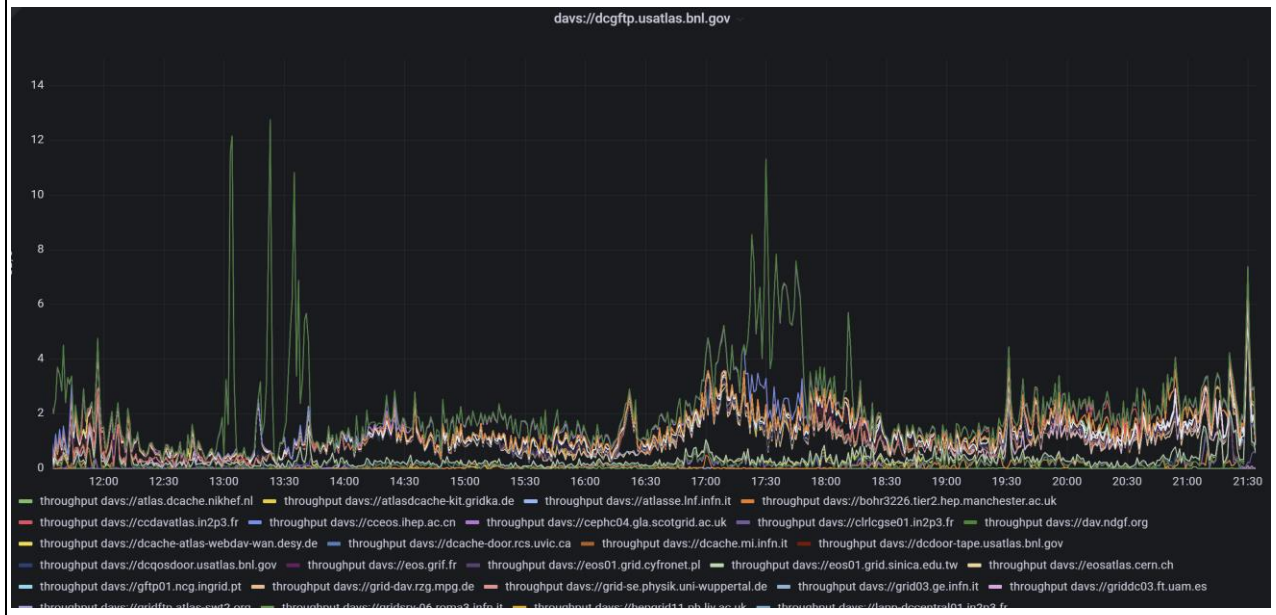
FTS table?

MySQL t_file table

DIV every row! 😯

```
SELECT
UNIX_TIMESTAMP(finish_time) DIV 60 * 60
AS "time",   sum(filesize) AS "throughput“ / 60,
dest_se
FROM t_file
WHERE  finish_time
BETWEEN FROM_UNIXTIME(XXXXX)
AND FROM_UNIXTIME(YYYYY)
and source_se='davs://dcgftp.usatlas.bnl.gov’
and  file_state='FINISHED’
GROUP BY 1, dest_se
ORDER BY 1;
```

MySQL can also do time series with some effort.



Brookhaven National Laboratory

FTS table?

MySQL t_file table

DIV every row! 😯

```
SELECT
UNIX_TIMESTAMP(finish_time) DIV 60 * 60
AS "time",   sum(filesize) AS "throughput“ / 60,
dest_se
FROM t_file
WHERE  finish_time
BETWEEN FROM_UNIXTIME(XXXXX)
AND FROM_UNIXTIME(YYYYY)
and source_se='davs://dcgftp.usatlas.bnl.gov’
and  file_state='FINISHED’
GROUP BY 1, dest_se
ORDER BY 1;
```

If it were **TimescaleDB**

```
Select
time_bucket(‘1m’, ‘finish_time’) as time,
sum(filesize) / 60 as throughput,
dest_se
from t_file
where
finish_time>’XXXX’ and
finish_time<‘YYYY’
and
source_se=‘davs://dcgftp.usatlas.bnl.gov’
and file_state=‘FINISHED’
group by 1, dest_se
order by 1
```

Brookhaven National Laboratory

Rucio DB at BNL

- Rucio DB has been stable for Belle II at BNL
- Rucio DB configuration can be improved for operation.
  - Wall Segment Size need to be larger.
  - Accumulation of history tables
    - Regular cleaning
    - Partitioning
- TimescaleDB
  - Time series DB
  - Auto-partitioning
  - SQL and not SQL like
  - Very fast aggregate functions over time