CBPM3 development

Antoine

CBPM meeting

April 10, 2020

– EPICS 7 and Python –

Next step

Create simple IOC/PV(s) that mimics the CBPM data structure:

- ✗ timestamp
- ✗ 4 button values
- ✗ bunch number
- ✗ ...else?

Run this IOC locally on my machine via EPICS 7

Update the IOC PV(s) either internally from C (would mimic the IOC running on a FPGA) or externally from Python (network bandwidth test) as fast as possible

Read-out the IOC from Python as fast as possible

Write information to database

Add more and more IOCs up to 120

Next step

Create simple IOC/PV(s) that mimics the CBPM data structure:

- ✗ timestamp
- ✗ 4 button values
- ✗ bunch number
- ✗ ...else?

Run this IOC locally on my machine via EPICS 7

Update the IOC PV(s) either internally from C (would mimic the IOC running on a FPGA) or externally from Python (network bandwidth test) as fast as possible

Read-out the IOC from Python as fast as possible

today's update

Write information to database

Add more and more IOCs up to 120

Simplest IOC I could manage

IOCs are still quite mysterious to me, especially how they are structured/written

I created an IOC with 5 records that increment at 10 Hz using the SCAN functionality (I did not write any C code to update the records):

```
record(calc, "$(user):timestamp")
{
        field(DESC, "timestamp")
        field(SCAN,".1 second")
        field(CALC, "(A<B)?(A+C):D")
        field(INPA, "$(user):timestamp.VAL  NPP NMS")
        field(INPB, "864000")
        field(INPC, "1")
        field(INPD, "0")
}

record(calc, "$(user):top_in")
{
        field(DESC, "top_in")
        field(SCAN,".1 second")
        field(CALC, "(A<B)?(A+C):D")
        field(INPA, "$(user):top_in.VAL  NPP NMS")
        field(INPB, "864000")
        field(INPC, "1")
        field(INPD, "0")
}

record(calc, "$(user):bot_in")
{
        field(DESC, "bot_in")
        field(SCAN,".1 second")
        field(CALC, "(A<B)?(A+C):D")
        field(INPA, "$(user):bot_in.VAL  NPP NMS")
        field(INPB, "864000")
        field(INPC, "1")
        field(INPD, "0")
}
```

1 timestamp and 4 button values that increment from 0 to 864,000 at 10 Hz (i.e. it takes a full day to wrap around the counter)

This IOC runs via EPICS on lnx6248:

```
(base) [atc93@lnx6248 test_ioc]$ ./st.cmd
#!../../bin/linux-x86_64/test_ioc
< envPaths
epicsEnvSet("IOC","test_ioc")
epicsEnvSet("TOP","/nfs/ilc/sim3/atc93/epics-7.0.3.1/test_ioc")
epicsEnvSet("EPICS_BASE","/nfs/cesr/online/acc_control/epics/base/v7")
cd "/nfs/ilc/sim3/atc93/epics-7.0.3.1/test_ioc"
## Register all support components
dbLoadDatabase "dbd/test_ioc.dbd"
test_ioc_registerRecordDeviceDriver pdbbase
## Load record instances
dbLoadTemplate "db/user.substitutions"
Substitution file error: syntax error
line 16: '}'
dbLoadRecords "db/test_iocVersion.db", "user=atc93"
filename="../dbStatic/dbLexRoutines.c" line number=264
No such file or directory dbRead opening file db/test_iocVersion.db
dbLoadRecords "db/dbSubExample.db", "user=atc93"
filename="../dbStatic/dbLexRoutines.c" line number=264
No such file or directory dbRead opening file db/dbSubExample.db
var mySubDebug 1
#traceIocInit
cd "/nfs/ilc/sim3/atc93/epics-7.0.3.1/test_ioc/iocBoot/test_ioc"
iocInit
Starting iocInit
############################################################################
## EPICS R7.0.3.1
## EPICS Base built Mar 27 2020
############################################################################
iocRun: All initialization complete
## Start any sequence programs
#seq sncExample, "user=atc93"
epics> dbl
atc93:timestamp
atc93:top_in
atc93:bot_in
atc93:bot_out
atc93:top_out
epics>
```

Read PVs from Python as fast as possible

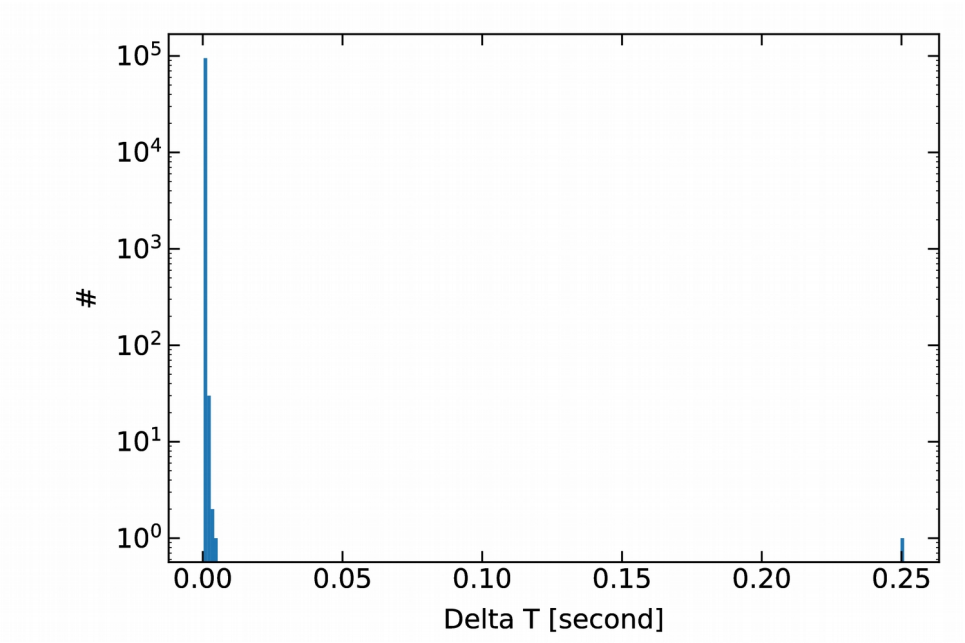
Using p4p Python module and via pvAccess, read as fast as possible the *.VAL PV of the 5 records (if record name is timestamp, a PV is for instance timestamp.VAL)

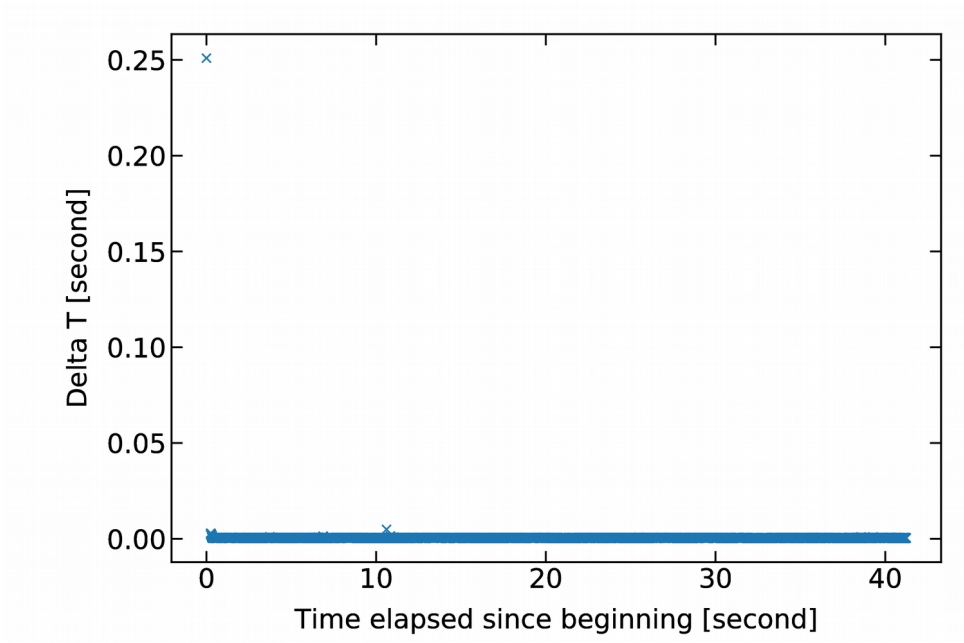
```
import time

import matplotlib.pyplot as plt
import numpy as np
from p4p.client.thread import Context

from python_custom_modules import mm_plot

ctxt = Context('pva')


def get():
    delta = []
    time_elapsed = []
    t0 = time.time()

    try:
        while True:
            t1 = time.time()
            time_elapsed.append(t1 - t0)
            timestamp, top_in, bot_in, bot_out, top_out = ctxt.get(
                ['atc93:timestamp', 'atc93:top_in', 'atc93:bot_in', 'atc93:bot_out', 'atc93:top_out'])
            delta.append(time.time() - t1)
    except KeyboardInterrupt:
        print(np.average(delta))
        fig, ax = mm_plot.create_figure('Delta T [second]', '#')
        ax.set_yscale('log')
        plt.hist(delta, bins=200)
        fig, ax = mm_plot.create_figure('Time elapsed since beginning [second]', 'Delta T [second]')
        plt.plot(time_elapsed[:len(time_elapsed) - 1], delta[:len(time_elapsed) - 1], 'x')
        plt.show()
```

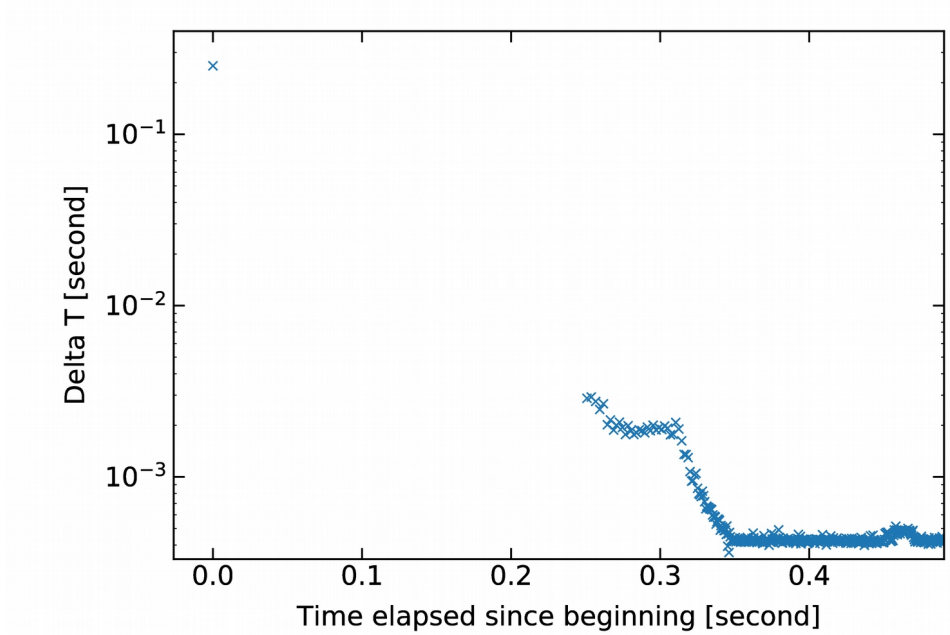
lines that has to do with the PV fetching

Read PVs from Python as fast as possible

For Python code also running on lnx6248: the typical read rate is ~2,000 Hz







first transactions are slower than the rest

Read PVs from Python using the monitor functionality

Monitor functionality → new read only when PVs updated in IOC

```python
def monitor():

    time_series = []
    delta_time = []

    def get_time(a):
        time_series.append(time.time())

    ctxt.monitor('atc93:timestamp', get_time)
    ctxt.monitor('atc93:top_in', get_time)
    ctxt.monitor('atc93:bot_in', get_time)
    ctxt.monitor('atc93:bot_out', get_time)
    ctxt.monitor('atc93:top_out', get_time)

    try:
        while True:
            time.sleep(0.0001)
    except KeyboardInterrupt:
        for i in range(len(time_series)-1):
            delta_time.append(time_series[i+1]-time_series[i])
        print(np.average(delta_time))
```

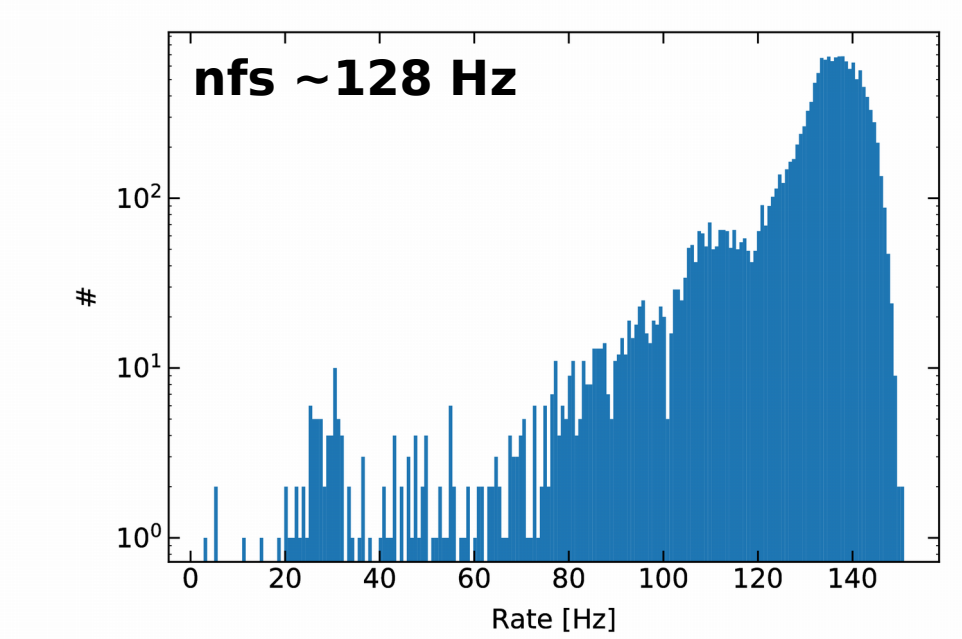
Measure pristine 10 Hz read rate which corresponds to the update rate of the PVs

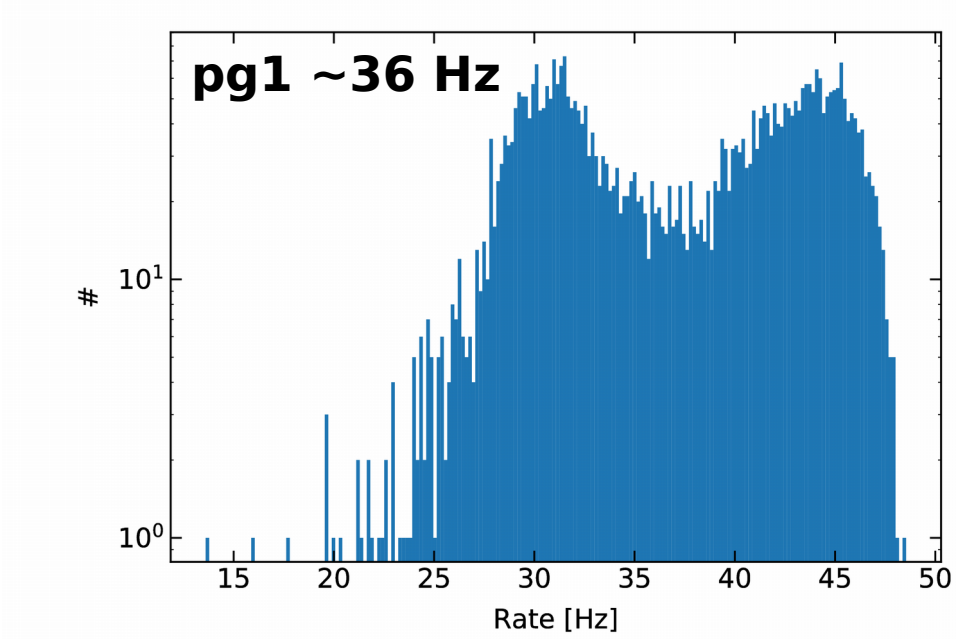
– PostgreSQL database –

nfs/ db versus pg1 db

Test: populating databases as fast as possible from lnx6248 (2-minute data set)

The nfs/ db service runs from lxn6248 (localhost)



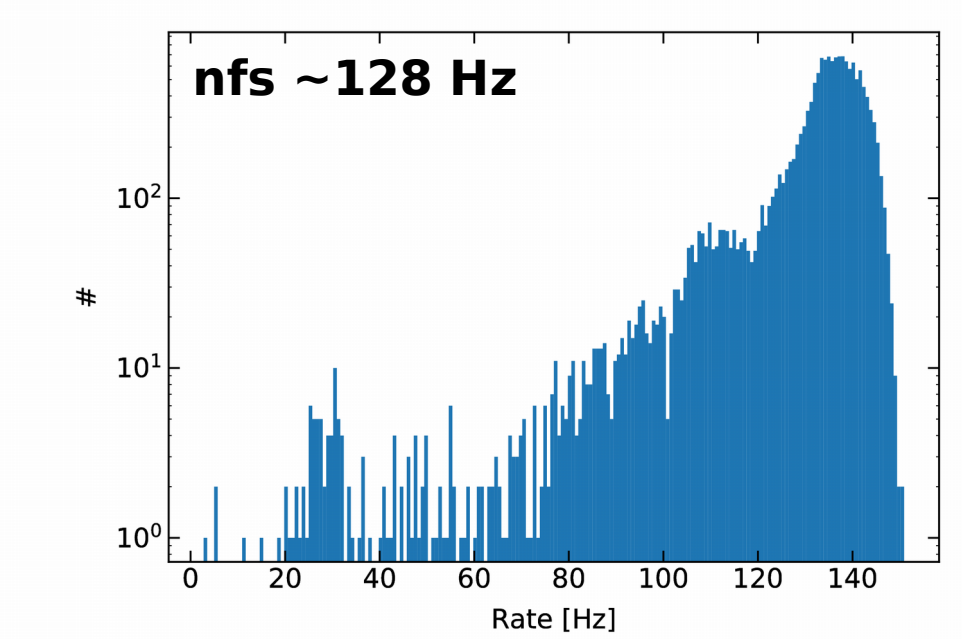


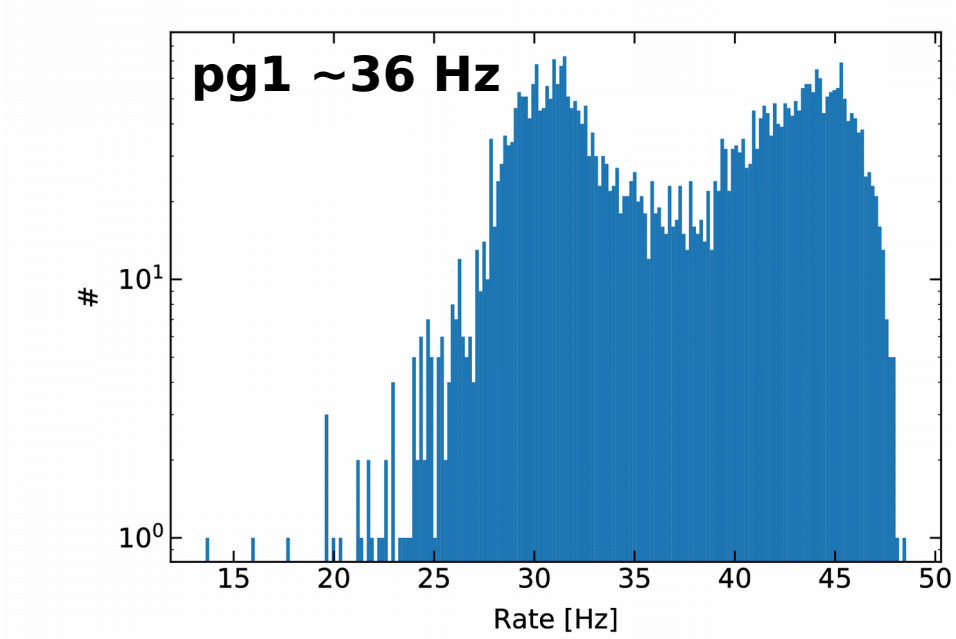
Running db service and populating code from same machine speed things up significantly (reduced network overhead)

nfs/ db versus pg1 db

Test: populating databases as fast as possible from lnx6248 (2-minute data set)

The nfs/ db service runs from lxn6248 (localhost)





Running db service and populating code from same machine speed things up significantly (reduced network overhead) → **IT group's expectation (Devin)**

Populating code: db commit

Test: compare versions of populating code to populate pg1 db from lnx6248

In the case of populating 120 tables, can:

- ✗ commit changes to db one table at a time
- ✗ commit changes to db for all the tables at once

	Rate [Hz]
one commit per table	~7
one commit for all tables	~35

→ **code default**

Populating code: db commit, sync vs async

Test: compare synchronous versus asynchronous commit **– Dan**

Synchronous ON: safe against system crash

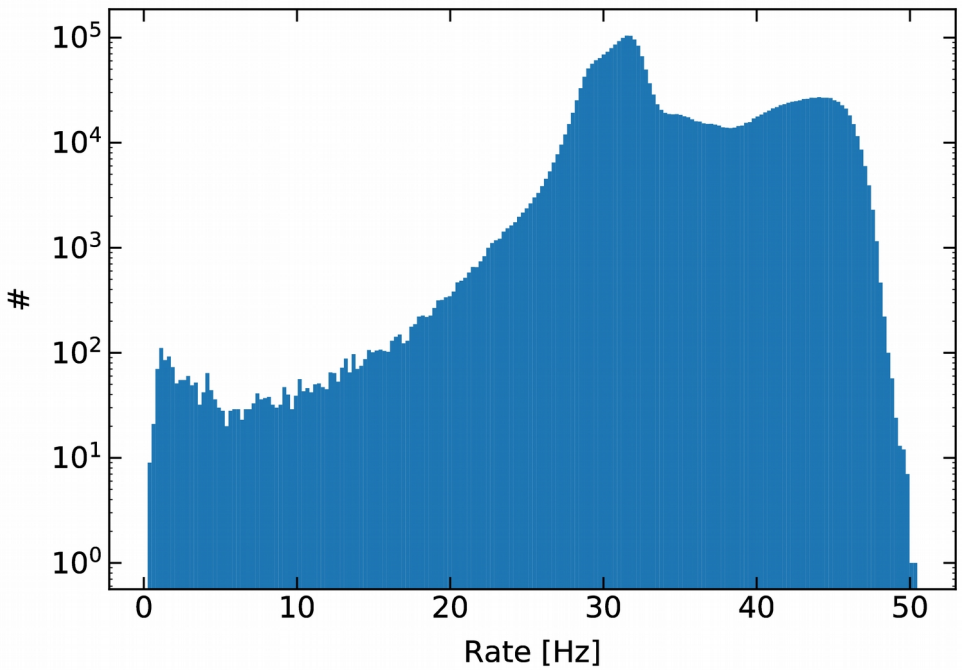
Synchronous OFF: fast but not safe against system crash (can loose commits) → "When off, there can be a delay between when success is reported to the client and when the transaction is really guaranteed to be safe against a server crash."

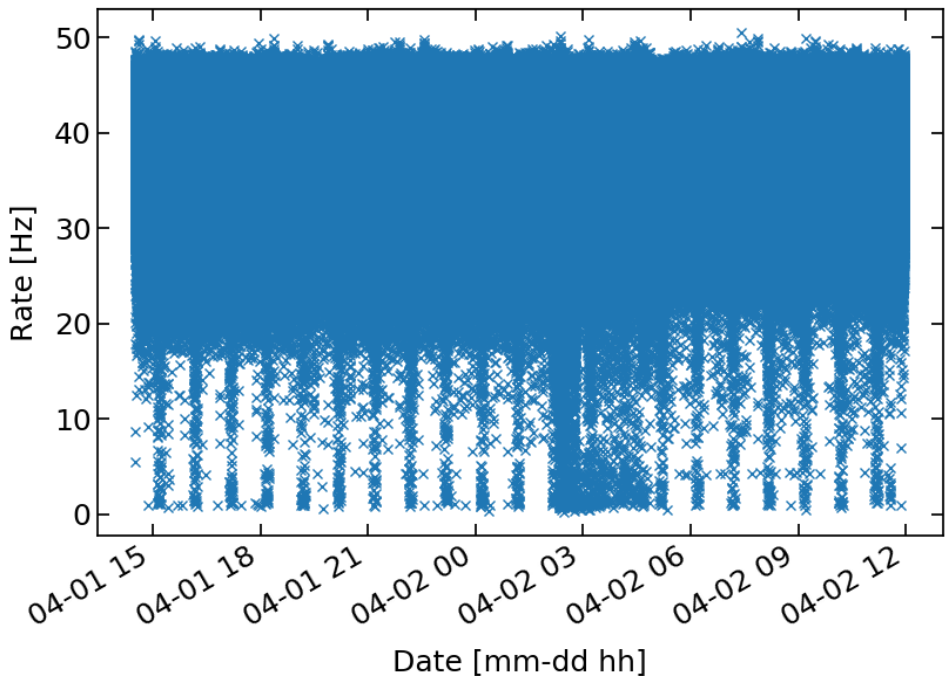
		Rate [Hz]
one commit per table	sync	~7
	async	~17
one commit for all tables	sync	~36
	async	~38.5

1-day run

Test: populate pg1 db from **lnx6248** for 1 day

→ clear ~hour time-scale pattern and early morning slow down

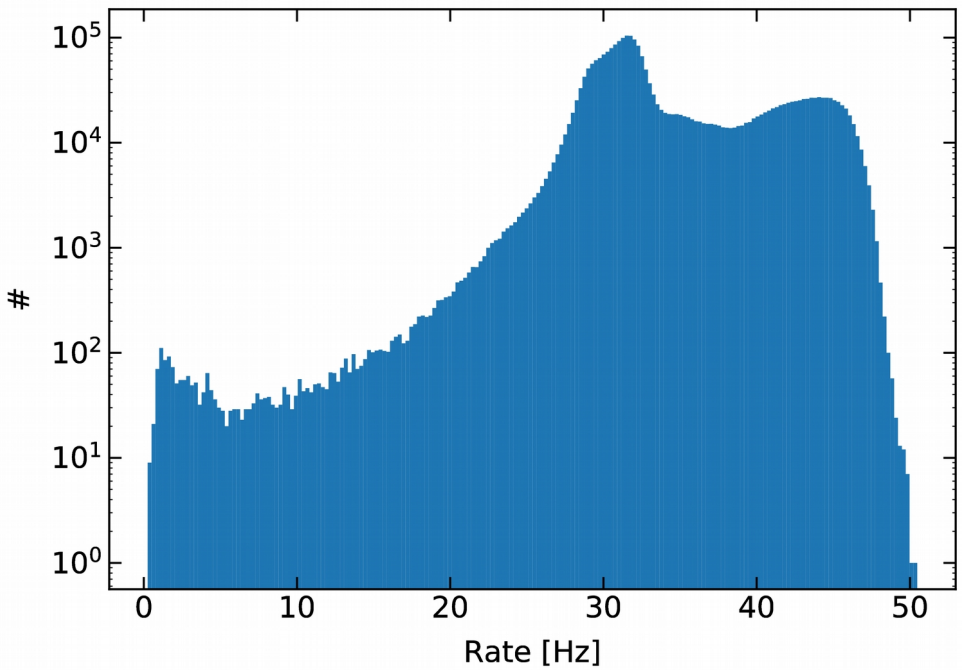


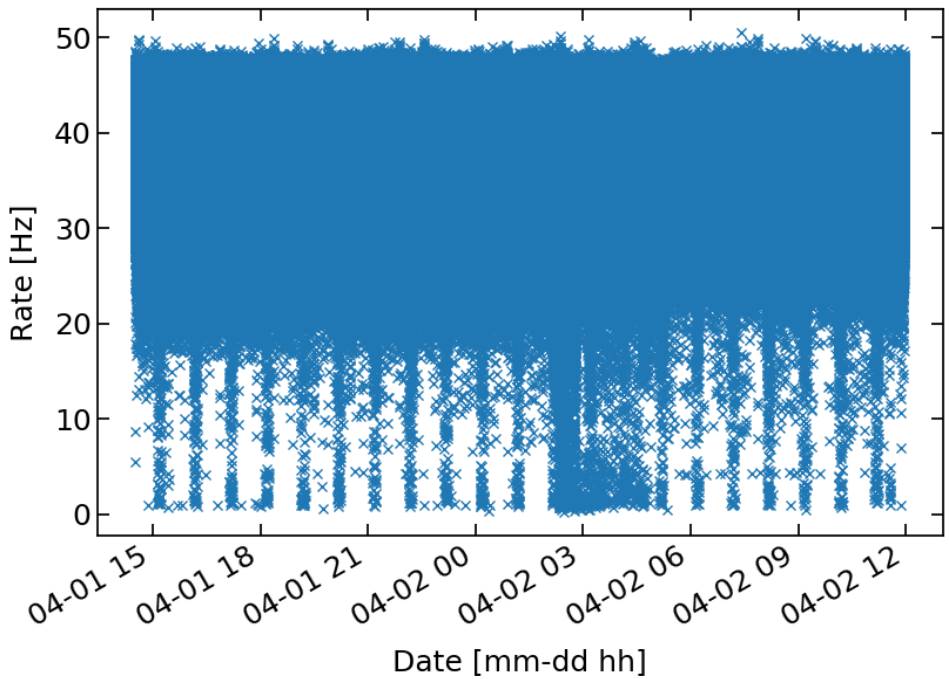


1-day run

Test: populate pg1 db from **lnx6248** for 1 day

→ clear ~hour time-scale pattern and early morning slow down





"There are cron jobs that run hourly, daily, and weekly that probably explain these. For the pg1 database server in particular, the nightly snapshot of the database (which is then backed-up to tape) started on 4/2 at 02:17:28." – **Devin**

PostgreSQL production server

IT group is setting up a “production CESR Postgres server” – **Devin**

The read/write code will run on “the actual node serving the CESR Postgres database” – **Devin**

– MongoDB –

MongoDB

“mongodb.classe.cornell.edu is a MongoDB 2.6.9 Server. We’re currently running MongoDB without authentication enabled, so for now I believe you can just connect, create your db, and proceed as needed. Eventually we will enable authentication, but for now we're waiting for an update to WebProtégé (which is also using mongodb).” – **Devin**

```
[atc93@lnx6248 atc93]$
[atc93@lnx6248 atc93]$ mongo
mongo   mongod  mongos
[atc93@lnx6248 atc93]$ mongo mongodb.classe.cornell.edu
MongoDB shell version v4.0.3
connecting to: mongodb://mongodb.classe.cornell.edu:27017/test
WARNING: No implicit session: Logical Sessions are only supported on server versions 3.6 and greater.
Implicit session: dummy session
MongoDB server version: 2.6.9
WARNING: shell and server versions do not match
Server has startup warnings:
2020-04-08T19:00:37.409-0400 [initandlisten]
2020-04-08T19:00:37.409-0400 [initandlisten] ** WARNING: You are running on a NUMA machine.
2020-04-08T19:00:37.409-0400 [initandlisten] **          We suggest launching mongod like this to avoid performance problems:
2020-04-08T19:00:37.409-0400 [initandlisten] **              numactl --interleave=all mongod [other options]
2020-04-08T19:00:37.409-0400 [initandlisten]
2020-04-08T19:00:37.409-0400 [initandlisten]
2020-04-08T19:00:37.409-0400 [initandlisten] ** WARNING: Readahead for /mnt/mongodb/data/db is set to 4096KB
2020-04-08T19:00:37.409-0400 [initandlisten] **          We suggest setting it to 256KB (512 sectors) or less
2020-04-08T19:00:37.409-0400 [initandlisten] **          http://dochub.mongodb.org/core/readahead
> db
test
> use actest
switched to db actest
> db
actest
> show collections
instr
system.indexes
>
```

Write to MongoDB

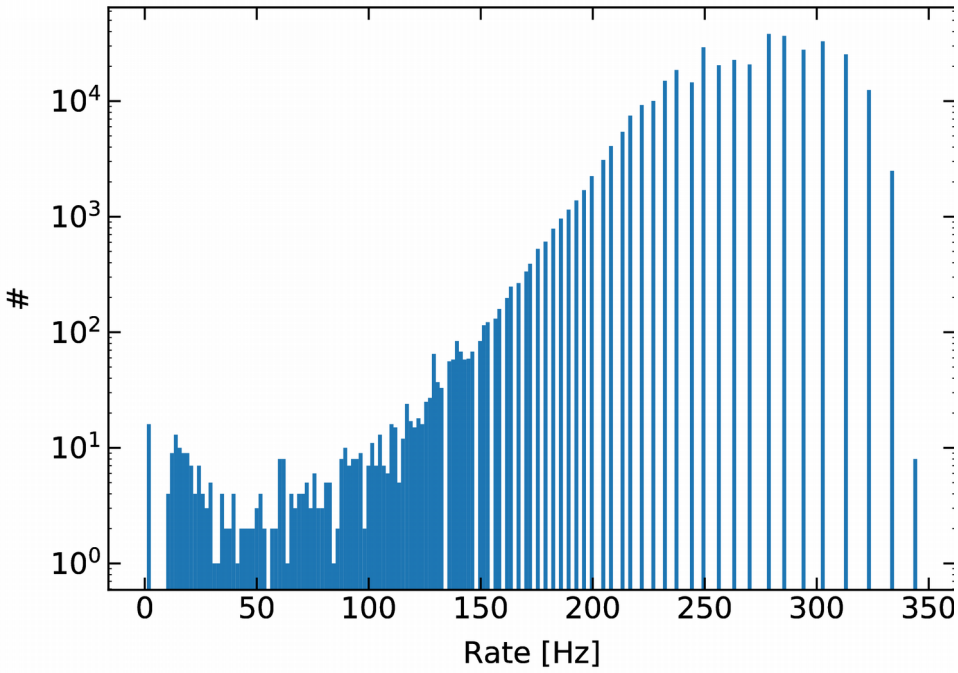
More straightforward to write the code than for PostgreSQL. Writing “data” from 120 CBPMs into one **collection** (equivalent to **table** in SQL world). Each **document** (equivalent to **row** in SQL world) has 1 timetsamp, 1 CBPM id, 4 button values:

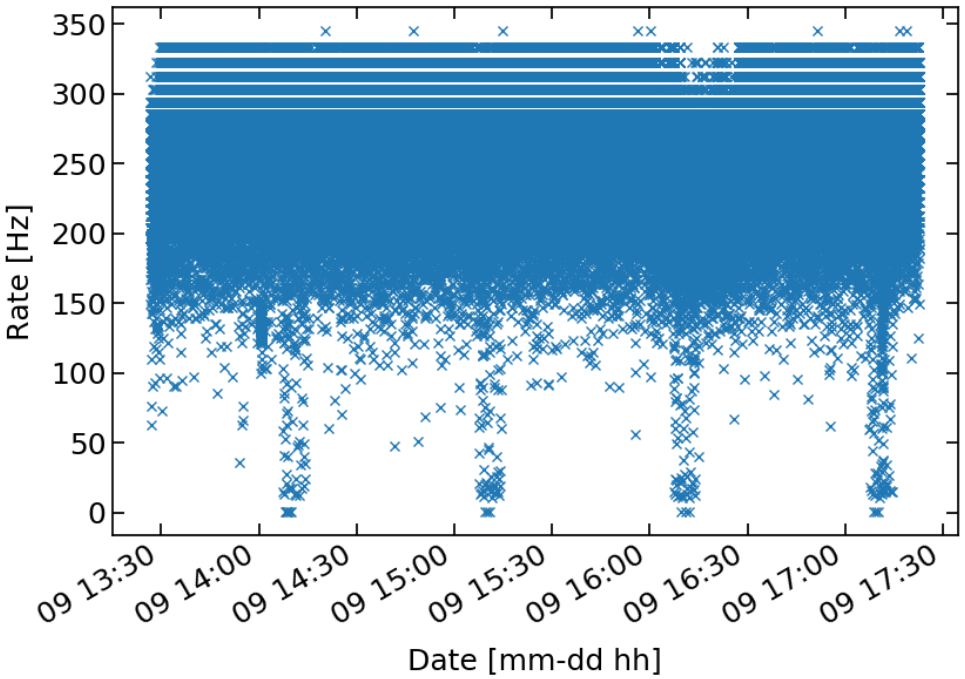
```
import datetime

from pymongo import MongoClient

# connect to DB service
client = MongoClient('mongodb://mongodb.classe.cornell.edu/')

# connect to DB
db = client['actest']

# delete collection
col_name = 'instr'
col = db[col_name]
col.drop()

# create/populate collections
while 1 == 1:
    timestamp = datetime.datetime.now()
    for i in range(120):
        col = db[col_name]
        post = {"timestamp": timestamp,
                "instr": str(i),
                "top_in": 1000,
                "bot_in": 1000,
                "bot_out": 1000,
                "top_out": 1000}
        col.insert_one(post)
```

did not try to optimize the code yet for performance

MongoDB write rate

Average write rate of ~260 Hz compared to ~35 Hz for PostgreSQL





"The network paths to all three should be very similar, since they are all in the same production cluster. The loads in individual cluster members do vary, though currently classedb (MariaDB) and mongodb are on the same node. PostgreSQL and MariaDB both have production loads on them, while MongoDB is currently only lightly used. The other probable factor is that the version of MongoDB we're running from Red Hat is fairly old, v2.6. Older versions of MongoDB sacrifice some consistency guarantees to get faster speed, they didn't implement full ACID consistency until later versions, so it's possible to get inconsistent query results that see partially completed transactions." – **Dan**

Additional materials