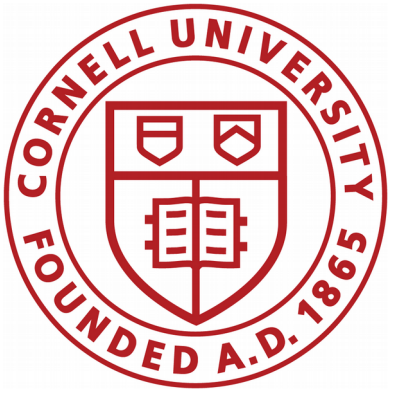


CBPM3 monitoring

Antoine

CBPM meeting

November 6, 2020

Database flush and snapshot

Design goal:

- ✗ keep 10 Hz granularity only for the last two weeks of data
- ✗ snapshot/save 1 trigger per minute for data older than two weeks
- ✗ flush remaining data older than two weeks

```
import psql_util
import datetime
import time

conn = psql_util.initialize()

while True:
    try:
        psql_util.flush_snapshot_cbpm3_data(conn)
        time.sleep(60)
    except:
        continue
```

Database flush and snapshot

Code flow:

- ✗ fetch timestamp corresponding to most recent cleanup
- ✗ fetch timestamp corresponding to most recent data
- ✗ extract time period requiring cleanup → split it in 1-minute slices
- ✗ loop through 1-minute slices:
  - ➢ fetch all the data for all the instruments
  - ➢ loop through instruments one bye one
    - ✔ extract first entry (timestamp/data)
    - ✔ delete from database all the data except first entry
  - ➢ write 1-minute slice timestamp to cleanup timestamp database

And it repeats, and it repeats...

Database flush and snapshot

Before flush/snapshot

timestamp	instr	top_in	bot_in	bot_out	top_out
2020-11-04 09:03:01.269367	10AE	0	0	0	0
2020-11-04 09:03:01.37153	10AE	1	1	1	1
2020-11-04 09:03:01.472524	10AE	2	2	2	2
2020-11-04 09:03:01.573873	10AE	3	3	3	3
2020-11-04 09:03:01.675357	10AE	4	4	4	4
2020-11-04 09:03:01.776814	10AE	5	5	5	5
2020-11-04 09:03:01.878432	10AE	6	6	6	6
2020-11-04 09:03:01.980056	10AE	7	7	7	7
2020-11-04 09:03:02.08171	10AE	8	8	8	8
2020-11-04 09:03:02.183172	10AE	9	9	9	9
2020-11-04 09:03:02.284525	10AE	10	10	10	10
2020-11-04 09:03:02.385844	10AE	11	11	11	11
2020-11-04 09:03:02.48741	10AE	12	12	12	12
2020-11-04 09:03:02.588941	10AE	13	13	13	13
2020-11-04 09:03:02.690649	10AE	14	14	14	14
2020-11-04 09:03:02.792018	10AE	15	15	15	15
2020-11-04 09:03:02.893684	10AE	16	16	16	16
2020-11-04 09:03:02.995256	10AE	17	17	17	17
2020-11-04 09:03:03.09668	10AE	18	18	18	18
2020-11-04 09:03:03.198351	10AE	19	19	19	19
2020-11-04 09:03:03.299602	10AE	20	20	20	20
2020-11-04 09:03:03.401349	10AE	21	21	21	21
2020-11-04 09:03:03.502779	10AE	22	22	22	22
2020-11-04 09:03:03.6044	10AE	23	23	23	23
2020-11-04 09:03:03.705874	10AE	24	24	24	24
2020-11-04 09:03:03.807635	10AE	25	25	25	25
2020-11-04 09:03:03.909209	10AE	26	26	26	26
2020-11-04 09:03:04.010806	10AE	27	27	27	27
2020-11-04 09:03:04.112316	10AE	28	28	28	28
2020-11-04 09:03:04.213713	10AE	29	29	29	29
2020-11-04 09:03:04.315207	10AE	30	30	30	30
2020-11-04 09:03:04.416588	10AE	31	31	31	31

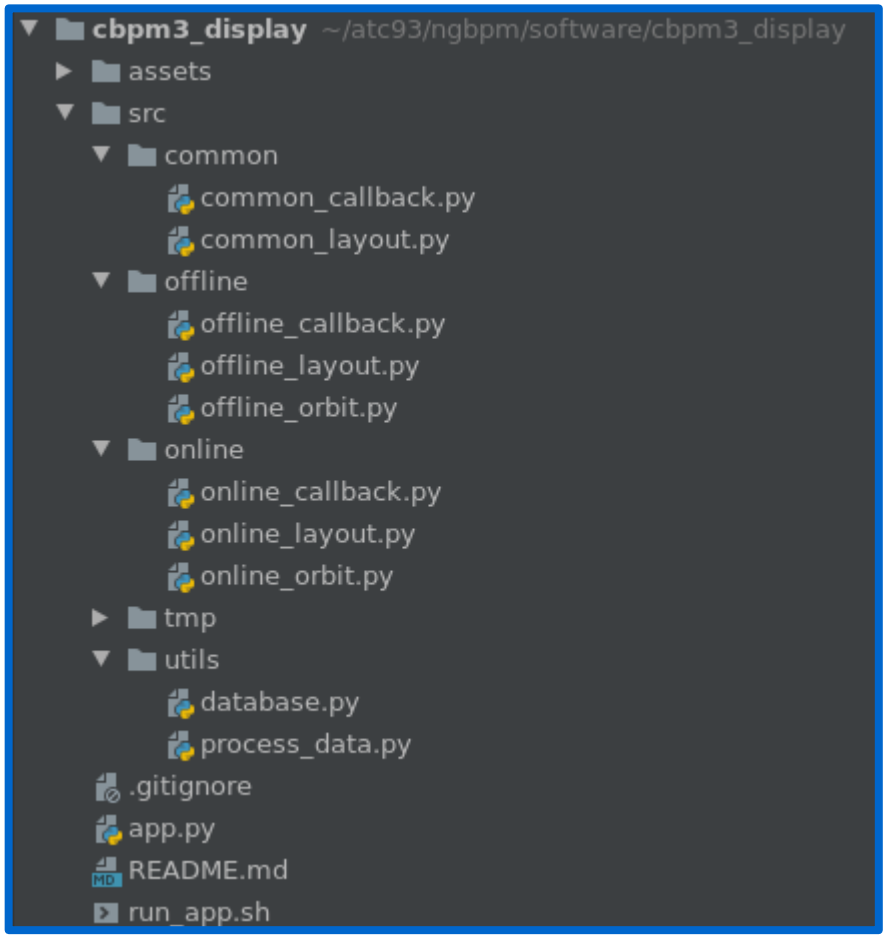
Database flush and snapshot

After flush/snapshot

timestamp	instr	top_in	bot_in	bot_out	top_out
2020-11-04 09:03:01.269367	10AE	0	0	0	0
2020-11-04 09:04:00.030802	10AE	579	579	579	579
2020-11-04 09:05:00.030795	10AE	1170	1170	1170	1170
2020-11-04 09:06:00.044652	10AE	1761	1761	1761	1761
2020-11-04 09:07:00.031649	10AE	2353	2353	2353	2353
2020-11-04 09:08:00.086051	10AE	2945	2945	2945	2945
2020-11-04 09:09:00.091946	10AE	3536	3536	3536	3536
2020-11-04 09:10:00.098064	10AE	4127	4127	4127	4127
2020-11-04 09:11:00.021353	10AE	4717	4717	4717	4717
2020-11-04 09:12:00.100264	10AE	5309	5309	5309	5309
2020-11-04 09:13:00.040077	10AE	5899	5899	5899	5899
2020-11-04 09:14:00.038121	10AE	6490	6490	6490	6490
2020-11-04 09:15:00.032579	10AE	7081	7081	7081	7081
2020-11-04 09:16:00.016937	10AE	7672	7672	7672	7672
2020-11-04 09:17:00.020153	10AE	8263	8263	8263	8263
2020-11-04 09:18:00.023302	10AE	8854	8854	8854	8854
2020-11-04 09:19:00.032528	10AE	9445	9445	9445	9445
2020-11-04 09:20:00.07367	10AE	10036	10036	10036	10036
2020-11-04 09:21:00.080125	10AE	10627	10627	10627	10627
2020-11-04 09:22:00.091164	10AE	11218	11218	11218	11218
2020-11-04 09:23:00.012452	10AE	11808	11808	11808	11808
2020-11-04 09:24:00.001413	10AE	12399	12399	12399	12399
2020-11-04 09:25:00.026527	10AE	12990	12990	12990	12990
2020-11-04 09:26:00.029044	10AE	13581	13581	13581	13581
2020-11-04 09:27:00.019171	10AE	14172	14172	14172	14172
2020-11-04 09:28:00.020145	10AE	14763	14763	14763	14763

Code re-organization/version control

Code is (finally) version controlled on GitHub. Until then, it lived on a disk that is backed-up by the IT group. For the occasion, and given most recent development, the code was once more re-organized (process was –almost– transparent).



Current effort: multi-session

In order to open multiple sessions, meaning having multiple web-browser tab opened or multiple users:

✗ need a mechanism for each session to have its own data

✗ data cannot be global and shared

> One of the core Dash principles explained in the Getting Started Guide on Callbacks is that **Dash Callbacks must never modify variables outside of their scope**. It is not safe to modify any `global` variables. This chapter explains why and provides some alternative patterns for sharing state between callbacks.

> In order to share data safely across multiple python processes, we need to store the data somewhere that is accessible to each of the processes.
>
> There are three main places to store this data:
>
> 1 - In the user's browser session
>
> 2 - On the disk (e.g. on a file or on a new database)
>
> 3 - In a shared memory space like with Redis

Current effort: multi-session

In order to open multiple sessions, meaning having multiple web-browser tab opened or multiple users:

✗ need a mechanism for each session to have its own data

✗ data cannot be global and shared

> One of the core Dash principles explained in the Getting Started Guide on Callbacks is that **Dash Callbacks must never modify variables outside of their scope**. It is not safe to modify any `global` variables. This chapter explains why and provides some alternative patterns for sharing state between callbacks.

> In order to share data safely across multiple python processes, we need to store the data somewhere that is accessible to each of the processes.
>
> There are three main places to store this data:
>
> 1 - In the user's browser session → implemented this solution
>
> 2 - On the disk (e.g. on a file or on a new database)
>
> 3 - In a shared memory space like with Redis

Previously on multi-session

Found network bottleneck for offline mode when using web-browser at home:

- ✗ each user (session) stores data in the HTML page
- ✗ if Python code and browser run on different machine → network data transfer

Running at home and asking for 10 minutes worth of data from the database resulted in waiting couple minutes to see the plots → not acceptable

Time decimation is mandatory to reduce the amount of data → for now I decimate from 0.1 second to 1 minute and performance looks reasonable. Will look if data compression is an option.

Current effort: multi-session

Time decimation helped tremendously. But having data going back and forth over the network is sub-optimal. Therefore, will try other solutions:

In order to share data safely across multiple python processes, we need to store the data somewhere that is accessible to each of the processes.

There are three main places to store this data:

1 - In the user's browser session

2 - On the disk (e.g. on a file or on a new database)

3 - In a shared memory space like with Redis

} solutions not requiring transfer over the network

Additional materials