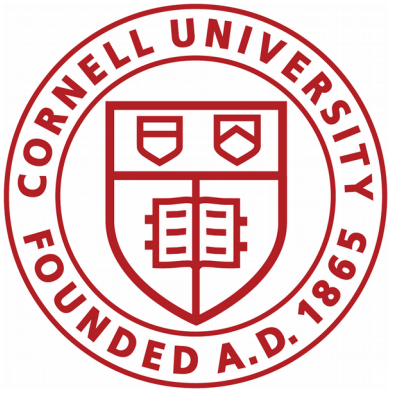

CBPM3 monitoring

Antoine

CBPM meeting

November 20, 2020

CBPM3 display needs

query data from DB

multi-user

decimate data

process data

display processed data

Current effort

Going from:

light-weight proof of concept

To:

full-scale proof of work

Light-weight proof of concept

All is working fine:

- ✗ query data from DB: between 30 min and 2 hours worth of data
- ✗ decimate data: from 0.1 second up to 1 minute interval
- ✗ process data: compute the x/y positions from individual buttons

All of the above lives in one *callback*:

> This chapter describes how to make your Dash apps using callback functions: Python functions that are automatically called by Dash whenever an input component's property changes.

and is cached/memoized so the output is saved in-memory (filesystem cache) and is made available code-wide per user (independent data for multi-session)

```
@cache.memoize()
def df_processing(start_date, end_date, time_select_min, time_select_max):
    df = db.get_offline_data_from_db_sqlalchemy(conn, start_date, end_date, time_select_min, time_select_max)
    df = analyze.position(df)
    df = analyze.decimate(df)
    return df
```

Full scale proof of work

All is not fine:

- ✗ query data from DB: 1 day worth of data
  - web-server (gunicorn) times out
  - virtual memory fills up
- ✗ decimate data: from 0.1 second up to 1 minute interval
  - web-server (gunicorn) times out

Timeout solution:

- ✗ set the timeout value to something very large → maybe asking for problems, plus a sign that the code is not designed well/optimized
- ✗ turn CPU and I/O intensive tasks into asynchronous (non-blocking) tasks

Virtual memory solution:

- ✗ query and decimate data in small batches

Asynchronous I/O tasks

Using out-of-the box co-routine libraries based on greenlet:

✗ gevent – compatible with psycopg2 PSQL Python module but also requires to install and setup psycogreen, see here)

✗ Eventlet – compatible with psycopg2 but only supports subset of PSQL queries

- The COPY query is not supportted :

```
psycopg2.ProgrammingError: copy_expert cannot be used with an asynchronous callback.
```

- The SELECT query is supported (implemented via sqlalchemy and pd.read_sql)

Using Eventlet could enable asynchronous DB query → no more time out issue! At the cost of 30% slower read-out rate.

So, an Eventlet approach could it seems solve the I/O blocking issue.

Asynchronous CPU tasks

If we solve the I/O issue we still have to solve the CPU one with the CPU intensive task of decimating the data for instance.

I devised a homemade tailored solution for having asynchronous processing with a somewhat complex approach and lots of handshaking. Not ideal...

So, instead of finding individual solutions for individual problems → why not finding an asynchronous solution that solves all our problems?

Introduction to Celery

- What's a Task Queue?
- What do I need?
- Get Started
- Celery is...
- Features
- Framework Integration
- Quick Jump
- Installation

What's a Task Queue?

Task queues are used as a mechanism to distribute work across threads or machines.

A task queue's input is a unit of work called a task. Dedicated worker processes constantly monitor task queues for new work to perform.

Celery communicates via messages, usually using a broker to mediate between clients and workers. To initiate a task the client adds a message to the queue, the broker then delivers that message to a worker.

A Celery system can consist of multiple workers and brokers, giving way to high availability and horizontal scaling.

It supports

- **Brokers**
  - RabbitMQ, Redis,
  - Amazon SQS, and more...
- **Concurrency**
  - prefork (multiprocessing),
  - Eventlet, gevent
  - thread (multithreaded)
  - *solo* (single threaded)
- **Result Stores**
  - AMQP, Redis
  - Memcached,
  - SQLAlchemy, Django ORM
  - Apache Cassandra, Elasticsearch, Riak
  - MongoDB, CouchDB, Couchbase, ArangoDB
  - Amazon DynamoDB, Amazon S3
  - Microsoft Azure Block Blob, Microsoft Azure Cosmos DB
  - File system
- **Serialization**
  - *pickle*, *json*, *yaml*, *msgpack*.
  - *zlib*, *bzip2* compression.
  - Cryptographic message signing.

Additional materials