A practical approach of different programming techniques to implement a real-time application using Django

Dipl.-Math. Sebastian Stigler sebastian.stigler@hs-aalen.de
Marina Burdack, MSc marina.burdack@hs-aalen.de
Aalen University of Applied Sciences, Germany

Motivation

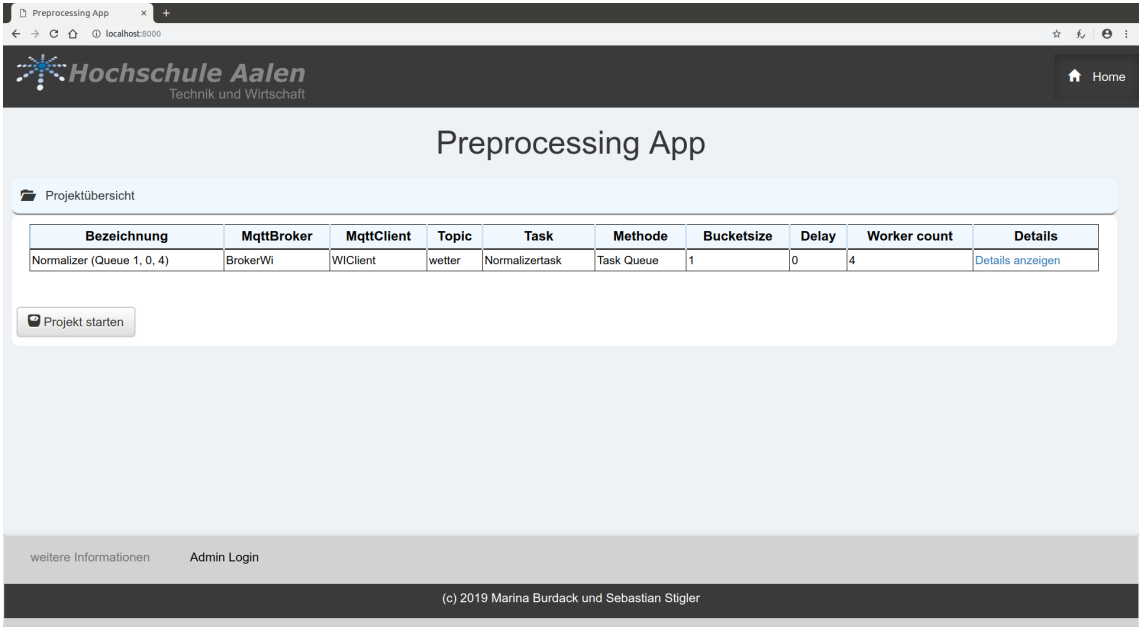
Aims for the Django Application

- How far do we get with an Python only approach?
- Tool to configure and run DA / ML pipeline
  - Datasource
  - Preprocessing Tasks
  - Machine Learning Tasks
  - Presentation of the Result

The Focus of this Paper

- The preprocessing part of the pipeline.
- How does our application scale?
- What are the knobs we can use to scale the application?

Preprocessing App (in german)

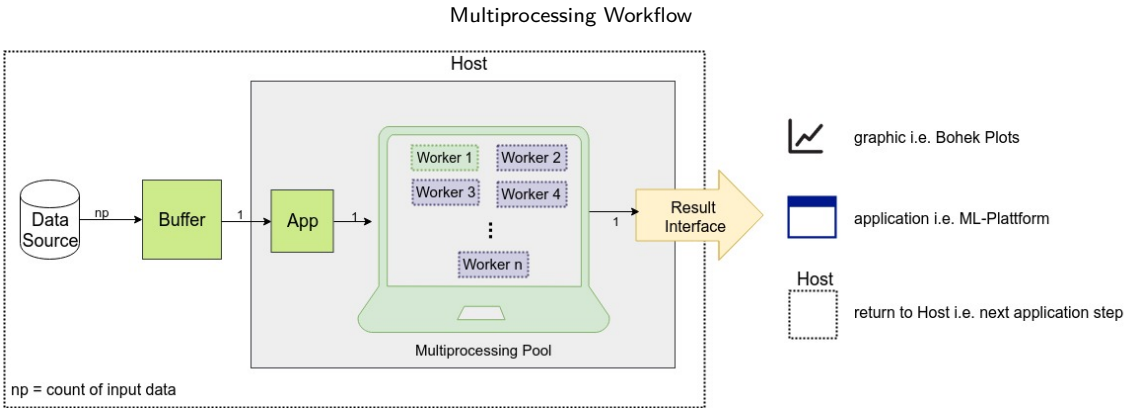


Source: own graphic

Methodology

Types of Scaling

- Singlethreaded ✗
- Multithreaded [4, 8] ✗
- Multiprocessing ✓
- Distributed Task Queue ✓

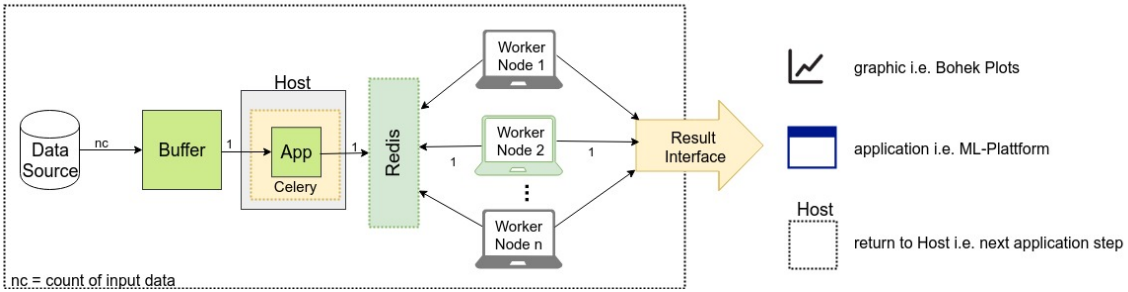


Source: own graphic

The Multiprocessing Pool is realized with the `ProcessPoolExecuter` Class of the `concurrent.futures` module [5] from Python 3.7's Standard Library.

Task Queue

Celery Workflow

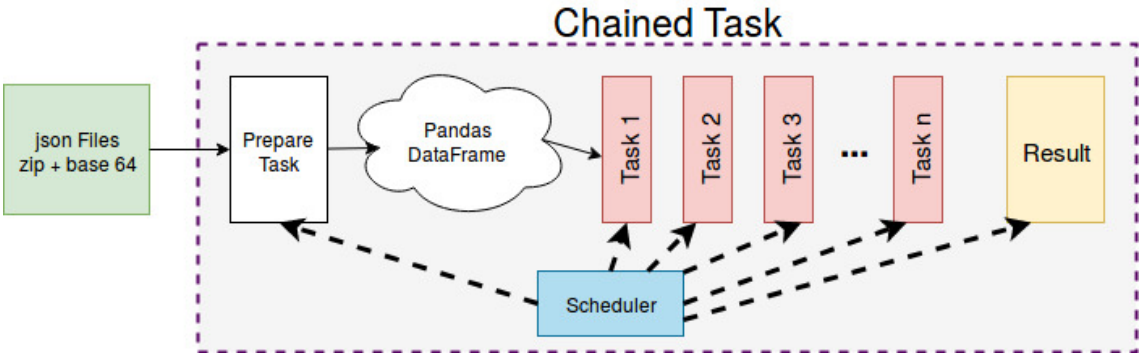


Source: own graphic

The Task Queue is realized with `Celery` 4.3 [7] and Redis [6].

Structure of a chained Task

Processing of a chained Task



Source: own graphic

Queueing Theory [1]

A queue with c servers is **stable** (won't grow without bound) if the following equation holds:

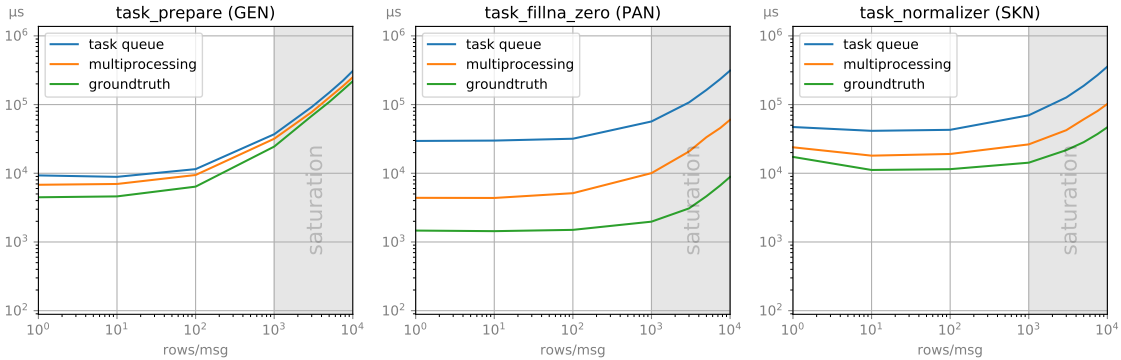
$$\rho = \frac{\lambda}{c\mu} < 1 \tag{1}$$

Where ρ is the server utilization, λ is the arrival rate and μ is the service rate (the inverse of the service time) for one task.

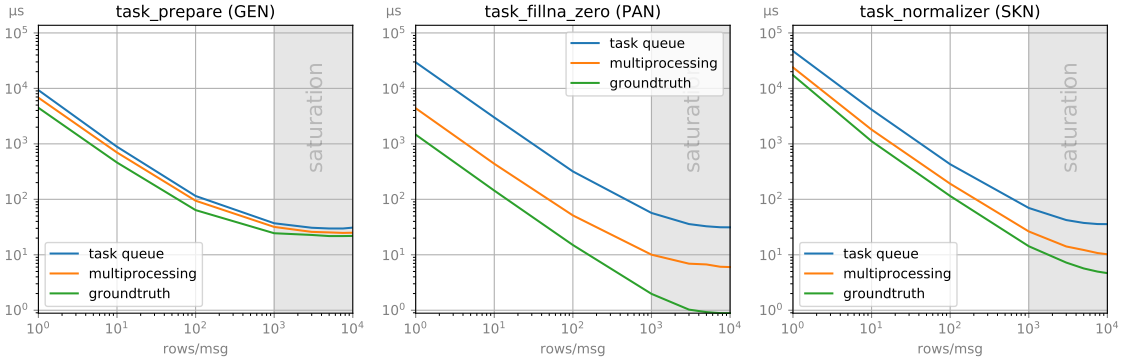
Evaluation

- 750'000 Measurements (***rows***) from a Davis Weatherstation
  - 33 value/row in total
  - 26 of them with numerical values
- 75 – 750'000 Messages (***msg***) are the output of the buffer with a rows/msg rate from 10000 down to 1
- 16 Subtasks
  - Prepare and Result Task
  - 6 Tasks which directly uses methods from Pandas [3]
  - 8 Tasks which uses preprocessing methods form scikit-learn [2]

Mean Servicetime per Message



Mean Servicetime per Row



Conclusion

- Python Libraries a sophisticated enough for scaling real-time applications.
- Buffering incomming datarows can compensate overhead for Task Queues.
- $\frac{\lambda}{\mu} < c$ determine's the scaling for the application.
- All results are applicable to the machine learning process too.

Thank you for your attention!

This was

A practical approach of different programming techniques to implement a real-time application using Django

Dipl.-Math. Sebastian Stigler sebastian.stigler@hs-aalen.de
Marina Burdack, MSc marina.burdack@hs-aalen.de

References I

[1] U. Narayan Bhat. *An Introduction to Queueing Theory. Modelling and Analysis in Applications*. Birkhäuser Basel, 2015. DOI: 10.1007/978-0-8176-8421-1.

[2] David Cournapeau and contriburors. *scikit-learn*. URL: https://scikit-learn.org.

[3] Wes McKinney et al. *Pandas. Python Data Analysis Library*. URL: https://pandas.pydata.org/.

[4] Python Software Foundation. *Thread State and the Global Interpreter Lock*. URL: https://docs.python.org/3/c-api/init.html#thread-state-and-the-global-interpreter-lock.

[5] Brian Quinlan. *concurrent.futures — Launching parallel tasks*. URL: https://docs.python.org/3/library/concurrent.futures.html.

[6] Salvatore Sanfilippo and contriburors. *Redis*. URL: hppts://redis.io.

[7] Ask Solem and contributors. *Celery: Distributed Task Queue*. URL: www.celeryproject.org.

[8] Thomas Wouters. *GlobalInterpreterLock*. URL: https://wiki.python.org/moin/GlobalInterpreterLock.