Taskfarm

A Client/Server Framework for Supporting Massive Embarrassingly Parallel Workloads

Magnus Hagdorn

Charité Universitätsmedizin Berlin

7th March 2024

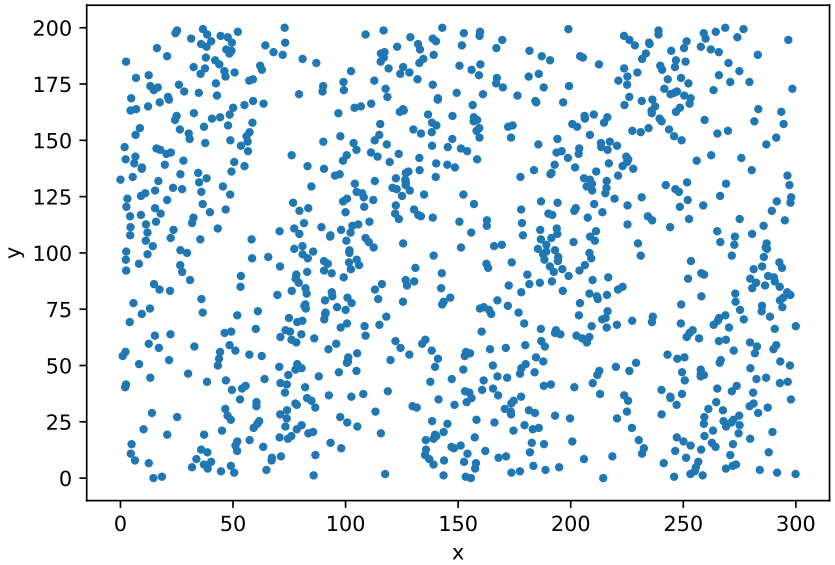
Outline

The Problem

- processing raw satellite data from ESA CryoSat mission
- data consist of point data in space and time along satellite paths
- create grid of ice elevation change

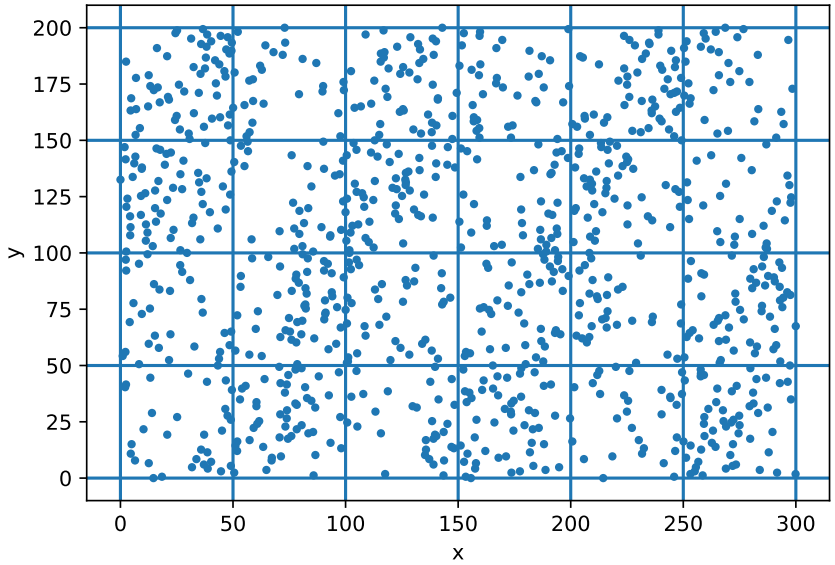
The Problem

Approach



The Problem

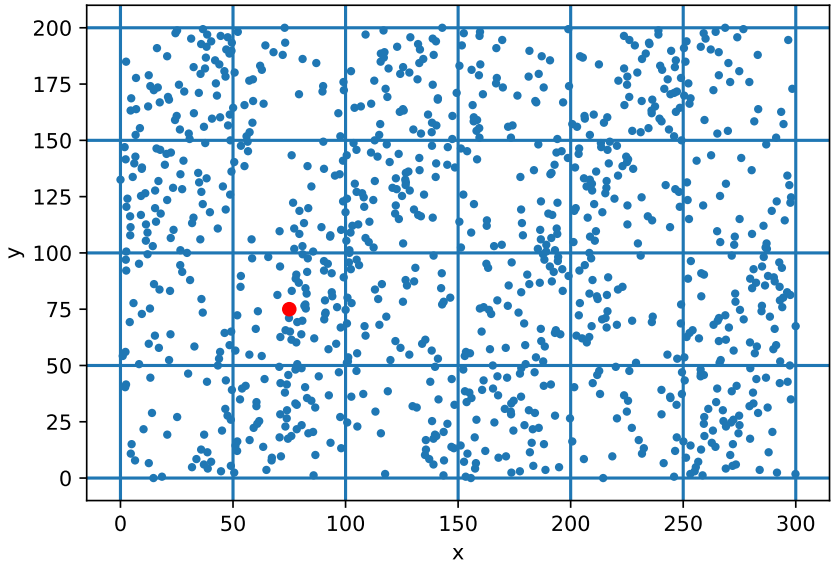
Approach



- define grid

The Problem

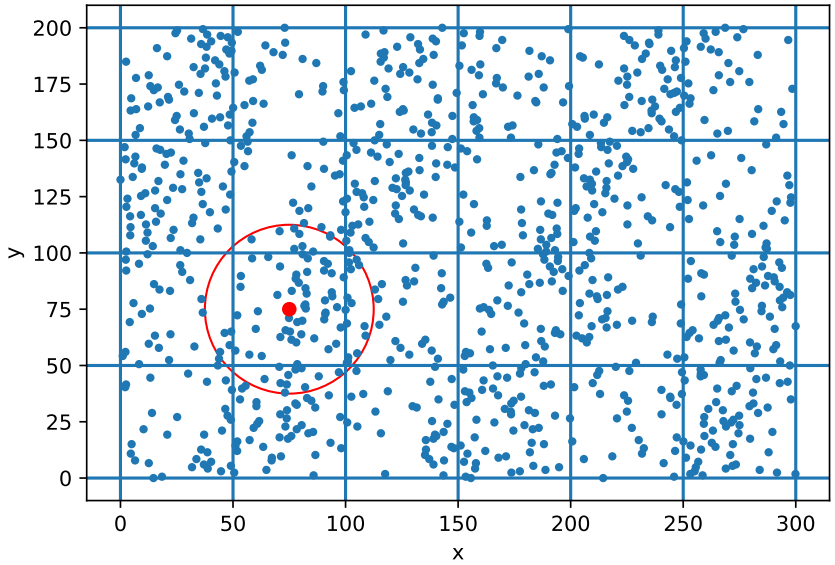
Approach



- define grid
- for each post

The Problem

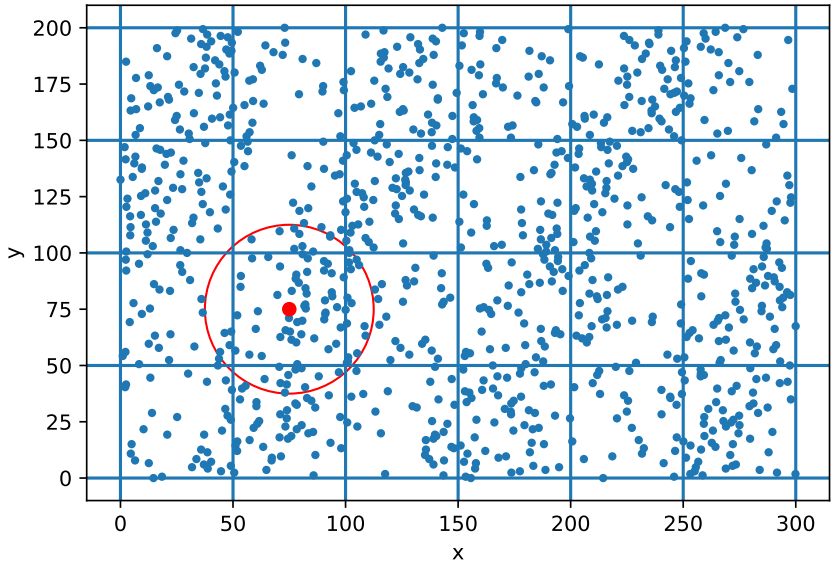
Approach



- define grid
- for each post
- consider all points that fall within radius r

The Problem

Approach

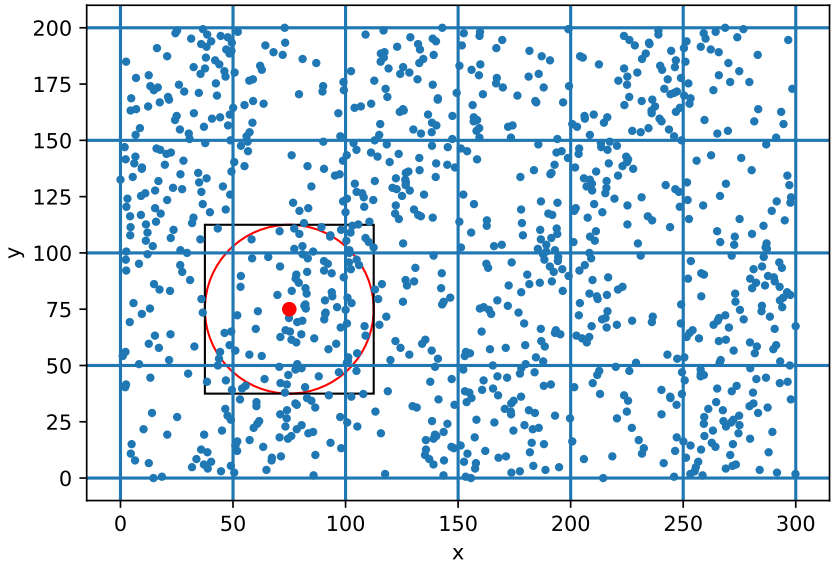


- define grid
- for each post
- consider all points that fall within radius r
- fit

$$z(x, y, t) = c_0 x + c_1 y + \dot{h} t + c_2$$

The Problem

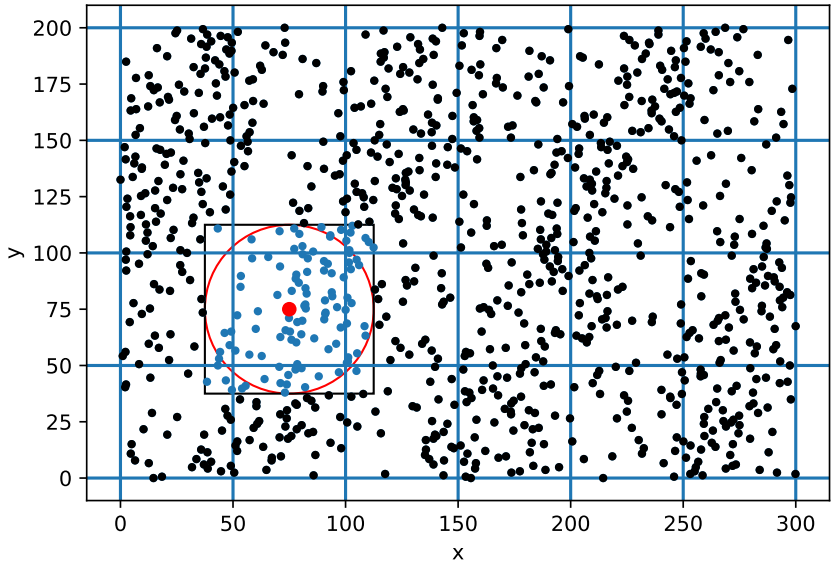
Approach



- consider points within square $2r$ centred on post

The Problem

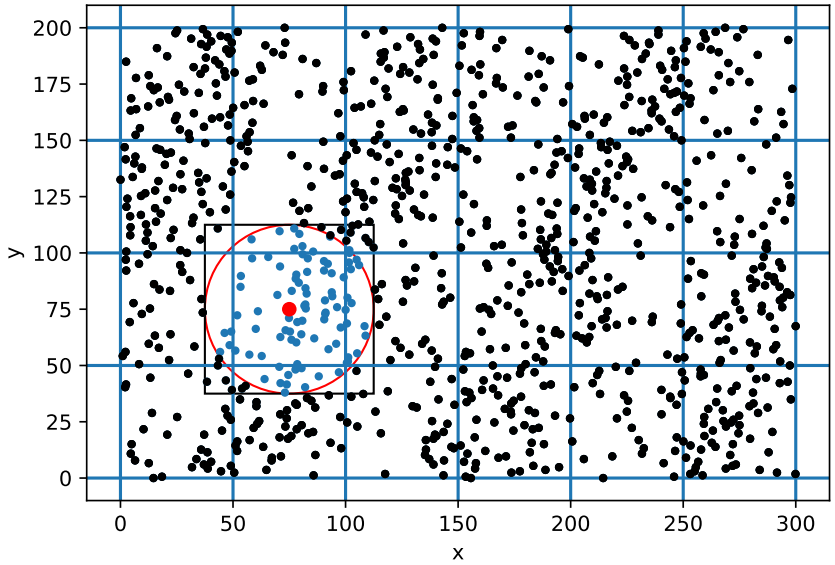
Approach



- consider points within square $2r$ centred on post
- calculate distance of each point to centre

The Problem

Approach



- consider points within square $2r$ centred on post
- calculate distance of each point to centre
- only use points with distance $< r$ for fitting

The Problem

Challenge

- big data, for Antarctica >1TB

The Problem

Challenge

- big data, for Antarctica >1TB
- lat/lon grid at poles has singularity

The Problem

Challenge

- big data, for Antarctica >1TB
- lat/lon grid at poles has singularity
- data set grows

Implementation

embarassingly parallel

Implementation

$$\text{embarassingly parallel} \Rightarrow \text{taskfarm}$$

Implementation

embarassingly parallel $\Rightarrow$ taskfarm

- define projection

Implementation

embarassingly parallel $\Rightarrow$ taskfarm

- define projection
- define output grid

Implementation

embarassingly parallel $\Rightarrow$ taskfarm

- define projection
- define output grid
- tile output grid

Implementation

embarassingly parallel $\Rightarrow$ taskfarm

- define projection
- define output grid
- tile output grid
- compute ice elevation change for each post

Implementation

embarassingly parallel $\Rightarrow$ taskfarm

- define projection
- define output grid
- tile output grid
- compute ice elevation change for each post
- assemble tiles

Implementation

Input

- load data, project it and dump into chunked point DB

Implementation

Input

- load data, project it and dump into chunked point DB
- point DB chunks are stored as pandas dataframes

Implementation

Input

- load data, project it and dump into chunked point DB
- point DB chunks are stored as pandas dataframes
- chunks are stored in deep directory structure

Implementation

Input

- load data, project it and dump into chunked point DB
- point DB chunks are stored as pandas dataframes
- chunks are stored in deep directory structure
- path is linearised chunk index (in hex)

Implementation

Output

- identify required chunks and load them

Implementation

Output

- identify required chunks and load them
- for each node

Implementation

Output

- identify required chunks and load them
- for each node
  - select points within square of size $2r$ (and other criteria)

Implementation

Output

- identify required chunks and load them
- for each node
  - select points within square of size $2r$ (and other criteria)
  - compute distance to centre and drop points $d > r$

Implementation

Output

- identify required chunks and load them
- for each node
  - select points within square of size $2r$ (and other criteria)
  - compute distance to centre and drop points $d > r$
  - fit data

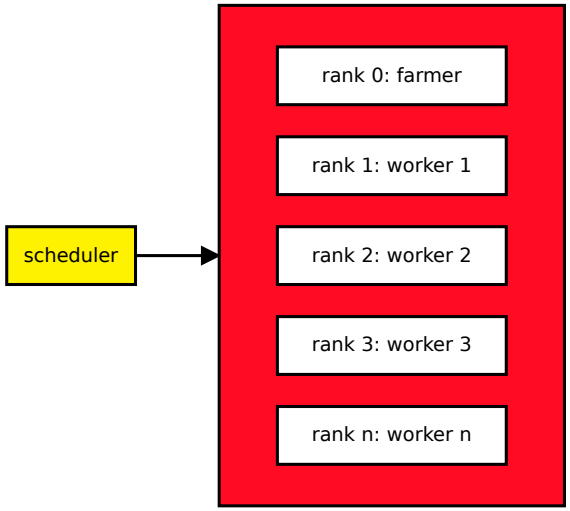
Implementation

Output

- identify required chunks and load them
- for each node
  - select points within square of size $2r$ (and other criteria)
  - compute distance to centre and drop points $d > r$
  - fit data
- store tile

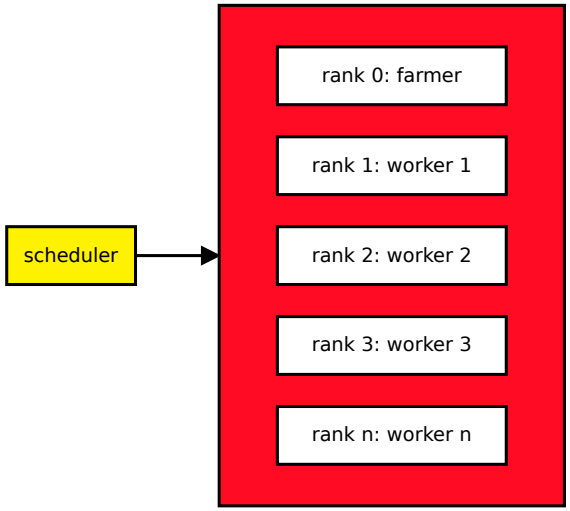
Taskfarm

MPI



Taskfarm

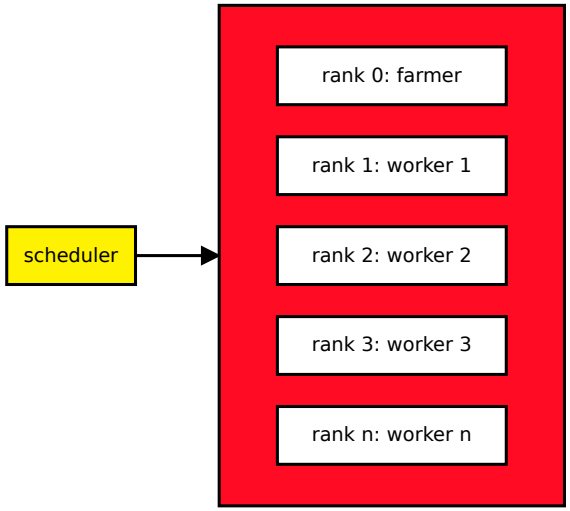
MPI



good a programming model we all know and love

Taskfarm

MPI

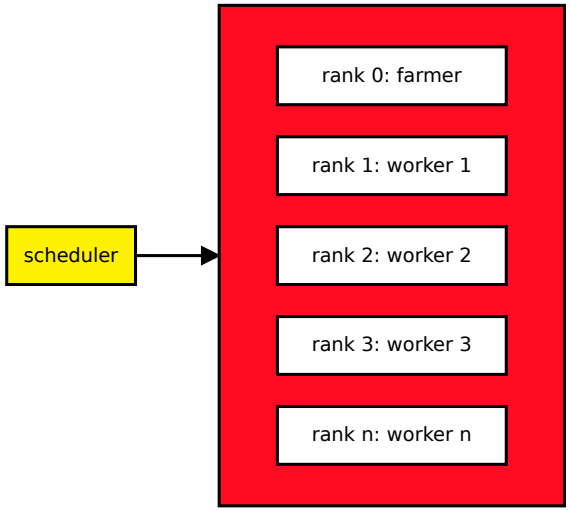


good a programming model we all know and love

bad the size of the task farm is fixed

Taskfarm

MPI



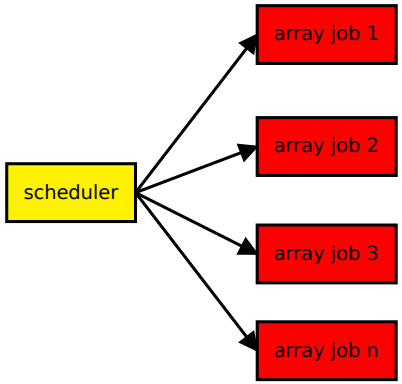
good a programming model we all know and love

bad the size of the task farm is fixed

ugly the entire farm burns down when a worker crashes

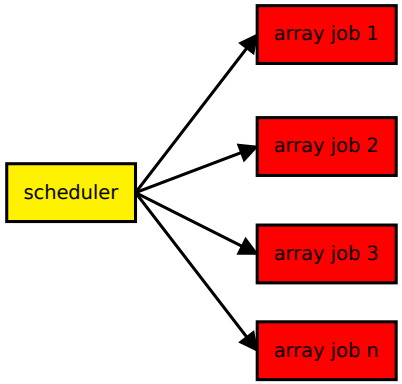
Taskfarm

Array Job



Taskfarm

Array Job

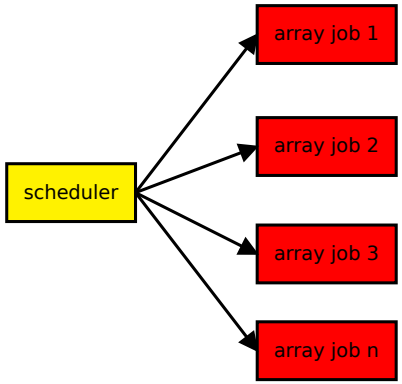


good

- simple
- the farm can dynamically grow/shrink
- crashing jobs don't take the farm down

Taskfarm

Array Job

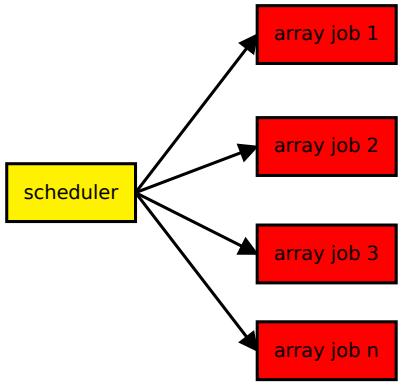


good
- simple
- the farm can dynamically grow/shrink
- crashing jobs don't take the farm down

bad small tasks take longer to schedule than run

Taskfarm

Array Job



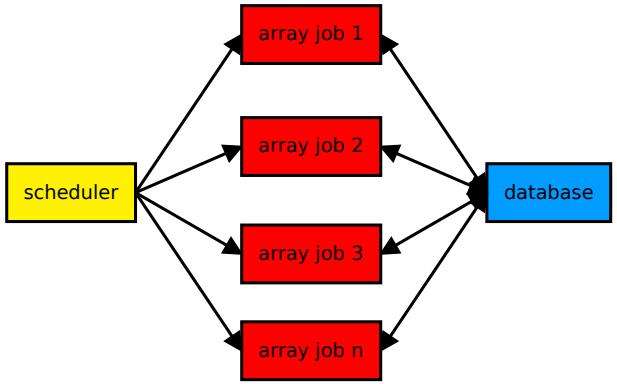
good
- simple
- the farm can dynamically grow/shrink
- crashing jobs don't take the farm down

bad small tasks take longer to schedule than run

ugly when the farm gets huge it is hard keeping track

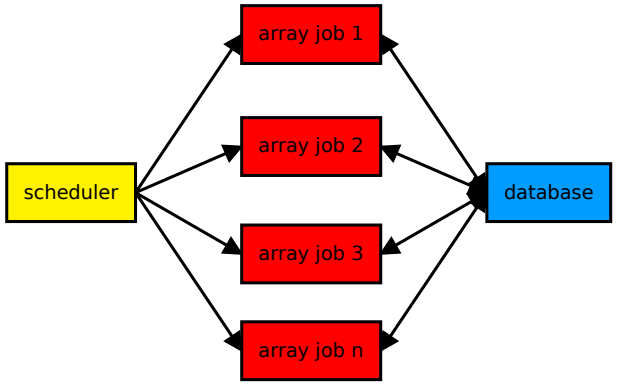
Taskfarm

Database



Taskfarm

Database

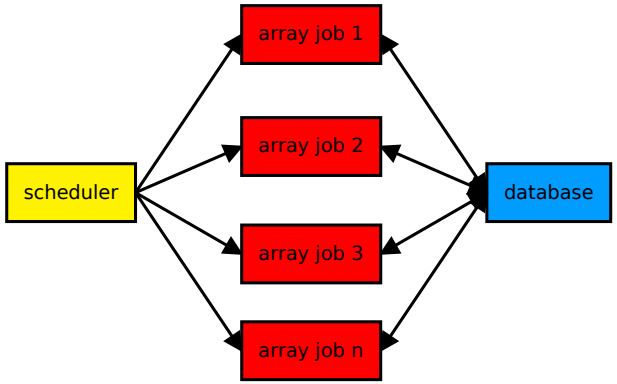


good

- state of farm is stored in DB
- can use auxiliary programs to monitor progress

Taskfarm

Database

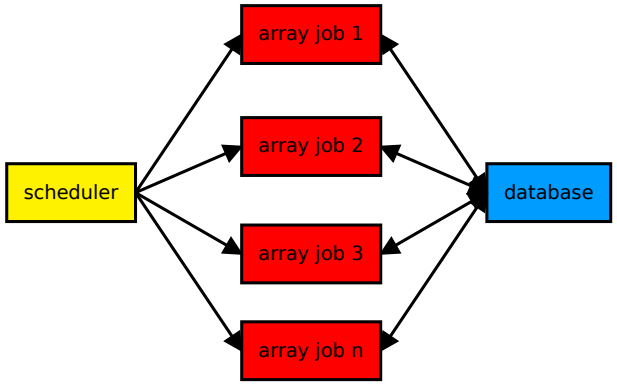


good
- state of farm is stored in DB
- can use auxiliary programs to monitor progress

bad need a proper DB with row locking

Taskfarm

Database



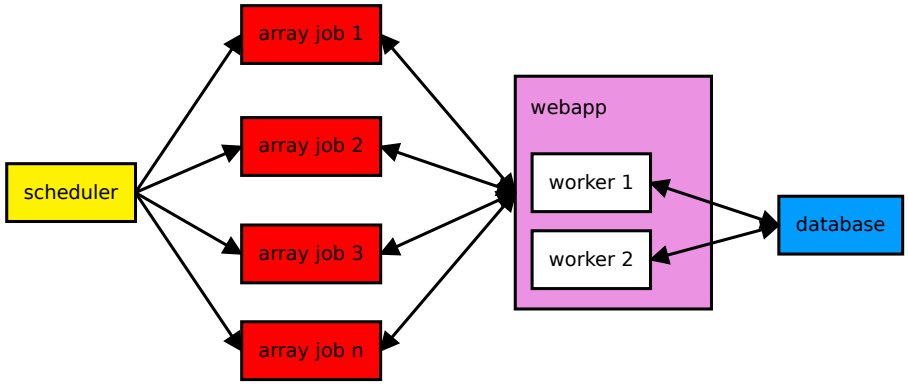
good
- state of farm is stored in DB
- can use auxiliary programs to monitor progress

bad need a proper DB with row locking

ugly limited number of parallel connections

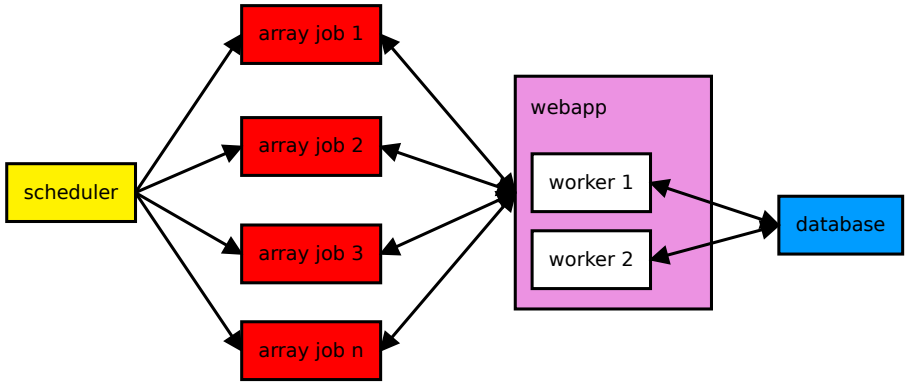
Taskfarm

Webapp



Taskfarm

Webapp

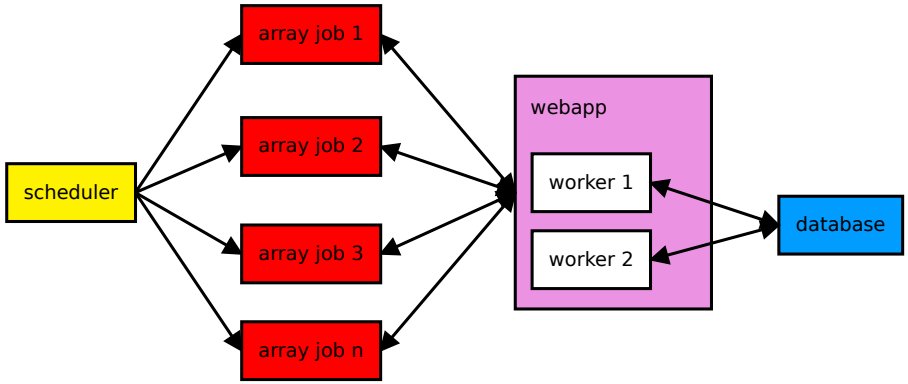


good

- can have many parallel connections
- can check taskfarm status from outside cluster

Taskfarm

Webapp



good
- can have many parallel connections
- can check taskfarm status from outside cluster

bad need to host the web app somewhere

Taskfarm

Implementation

- flask
- flask-sqlalchemy
- flask-httpauth
- a database with row locking, eg postgresgl
- gunicorn
- web server

Taskfarm

Example Code

```
from taskfarm_worker import TaskFarmWorker

# connect to taskfarm
tf = TaskFarmWorker(
     'user', 'secret',
     'da8eb1c10eac4cefb39c8889d6d7170a',
     url_base='http://localhost:5000/api/')
print(tf.percentDone)

# loop over the tasks
for t in tf.tasks:
     print("worker {} processing task {}"
           .format(tf.worker_uuid, t))
     # do some work
     # update the percentage done
     tf.update(50)
     # do some more work and update percentage
     tf.update(100)
     # mark task as completed
     tf.done()
```

Software Publishing

- documentation - sphinx + read the docs

Software Publishing

- documentation - sphinx + read the docs
- unit testing of both server and client

Software Publishing

- documentation - sphinx + read the docs
- unit testing of both server and client
- CITATION.cff

Software Publishing

- documentation - sphinx + read the docs
- unit testing of both server and client
- CITATION.cff
- submitted to Journal of Open Research Software

Software Publishing

- documentation - sphinx + read the docs
- unit testing of both server and client
- CITATION.cff
- submitted to Journal of Open Research Software
- docker container with server

Software Publishing

- documentation - sphinx + read the docs
- unit testing of both server and client
- CITATION.cff
- submitted to Journal of Open Research Software
- docker container with server

DOI: 10.5334/jors.393