Code Profiling

Kamilla Kopec-Harding

2022-11-02

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

Introduction

One of the main performance limitations of this tool is the slow calculation speed.

End to end, analysis of a single dam takes a minimum of ~ 16 min, consisting of (tested using GawLan (id=12)):

- [1.0 M] Initialization and upload
- [10.5 M] GEE Analysis
- [3.5 M] Post-processing (exporting, downloading and checking results)

Python Profiling

A number of tools are available for profiling python code including timeit and cProfile. They are discussed in the following resources:

- https://machinelearningmastery.com/profiling-python-code/
- https://stackoverflow.com/questions/582336/how-do-i-profile-a-python-script
- https://towardsdatascience.com/how-to-profile-your-code-in-python-e70c834fad89
- https://www.toucantoco.com/en/tech-blog/python-performance-optimization
- https://www.geeksforgeeks.org/profiling-in-python/

Unfortunately, due to the delayed execution model of Google Earth Engine, these can only be used to investigate the parts of the code that runs locally.

It is unlikely that major bottlenecks within the code will be found in these parts of the code.

Earth Engine Profiling

In order to find bottlenecks in the pipeline to target for improvement, we need to profile the code that runs on GEE.

GEE has a number of builtin code profiling tools discussed below:

- https://philippgaertner.github.io/2017/10/profiler/
- https://gis.stackexchange.com/questions/379634/optimize-earth-engine-script-script-profiler

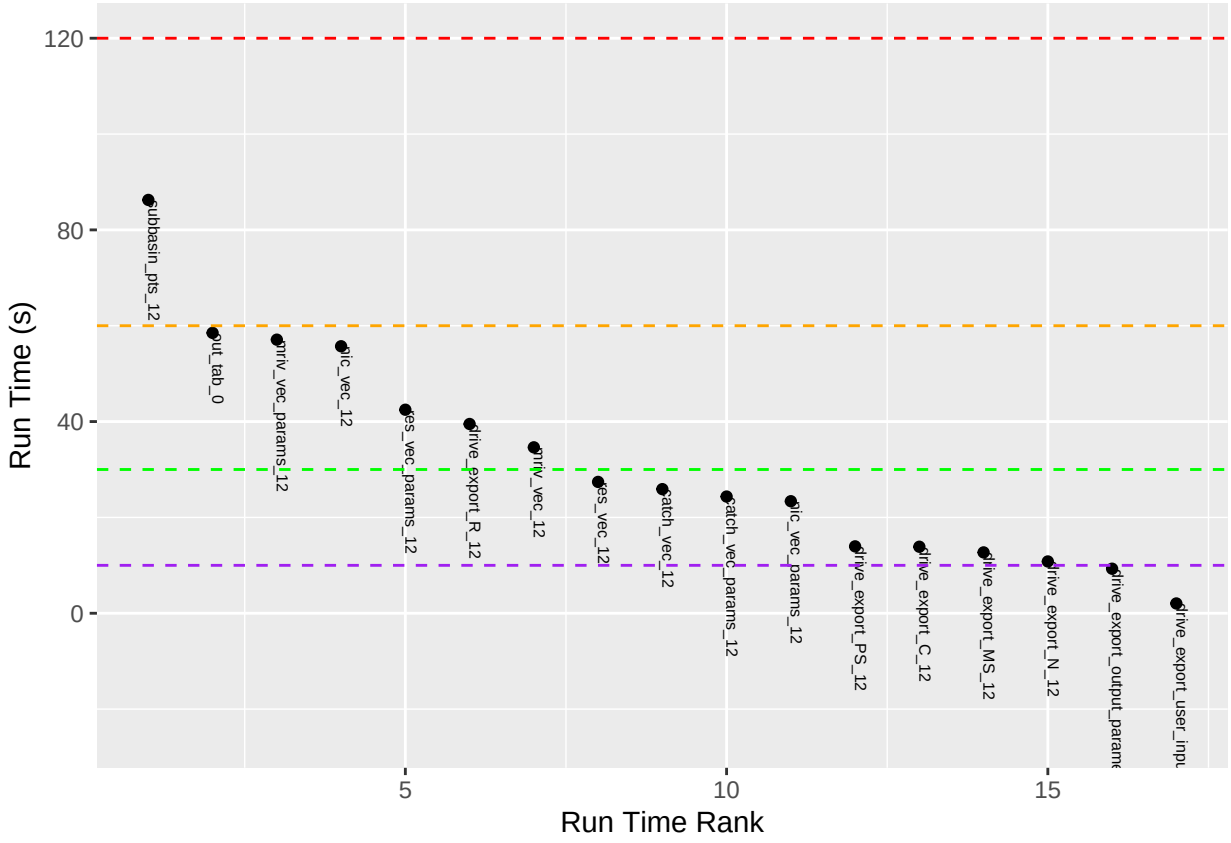
However, GEE's built-in code profiling tools only work in the interactive code-editor with javascript.

If we want to use these, we need to identify and prioritise parts of the code to translate to js and look at more closely.

We can do this by examining run-times and queue times of each pipeline step via the GEE task list.

Analysis of GEE Task List

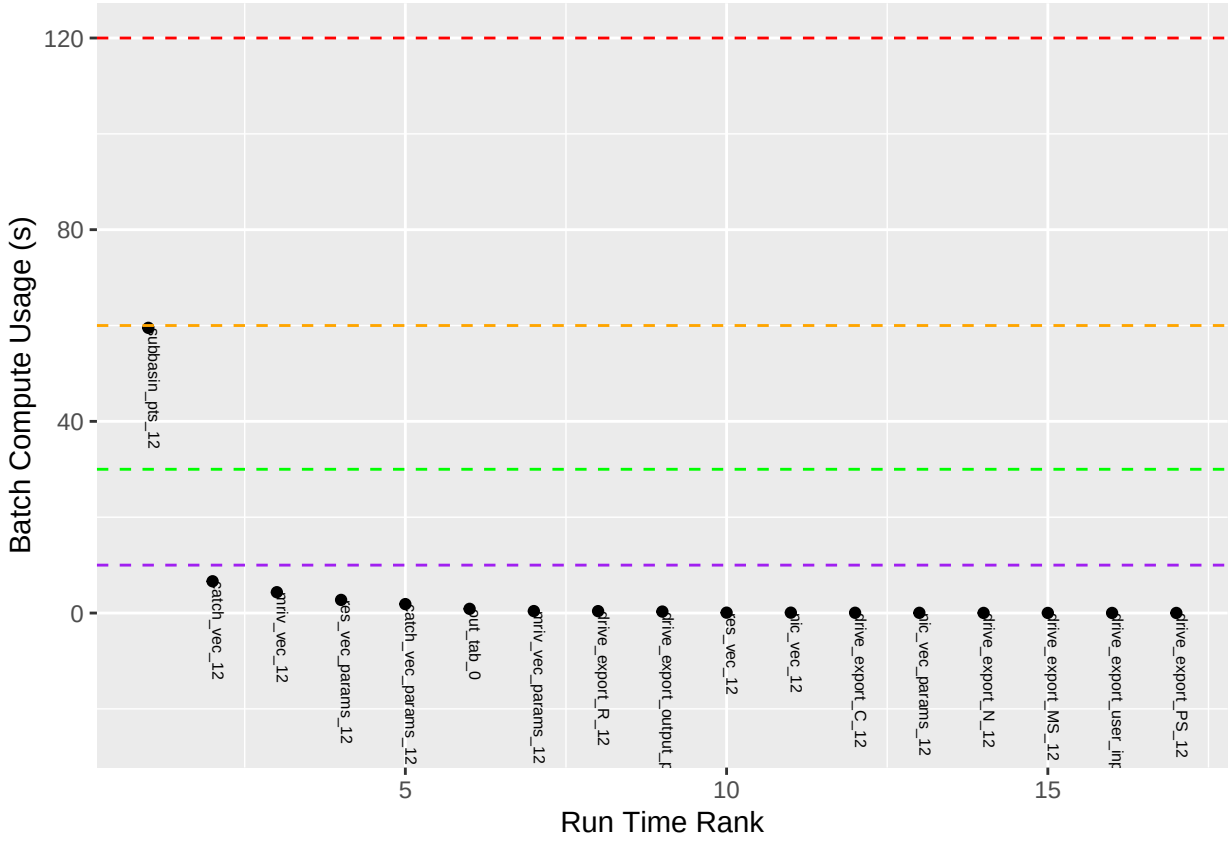
We analysed the pipeline run-times for analysis of GawLan (batch size 1). This is what we found.



- Each step takes between 1 and 1482 seconds (mean: 40 s).
- The longest step is XHEET-X002-12 Snapped dam locations and upstream basins:
- The following tasks take longer than 30s and should be targeted for improvement:

```
longer_than_30s
```

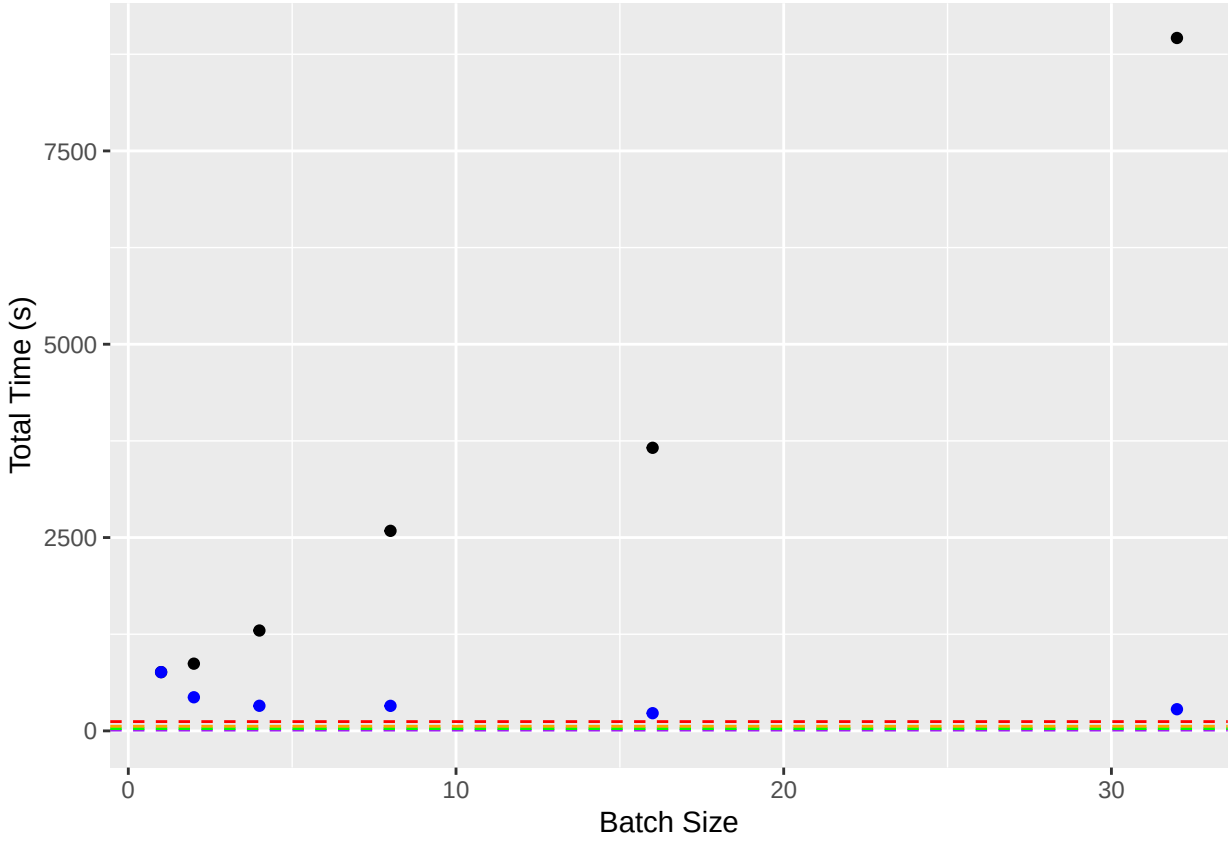
```
## # A tibble: 7 x 2
##   fmt_run_time description
##   <Period>     <chr>
## 1 1M 26.253S   XHEET-X002-12 Snapped dam locations and upstream basins
## 2 58.51S       XHEET-X021-0 All output parameters
## 3 57.096S      XHEET-X018-12 Params main river vector
## 4 55.72S       XHEET-X012-12 Non-inundated catchment vector
## 5 42.473S      XHEET-X011-12 Params reservoir vector
## 6 39.496S      XHEET-X011-0 Exporting to drive R_12
## 7 34.626S      XHEET-X017-12 Main river vector
```



- Each step takes between 0.01 and 120.40 ecu seconds (mean: 6.30 ecu s).
- The longest step is: XHEET-X002-12 Snapped dam locations and upstream basins.
- The following tasks take longer than 30s and should be targeted for improvement:

```
## # A tibble: 1 x 2
##   batch_eecu_usage_seconds description
##                      <dbl> <chr>
## 1                     59.5 XHEET-X002-12 Snapped dam locations and upstream bas~
```

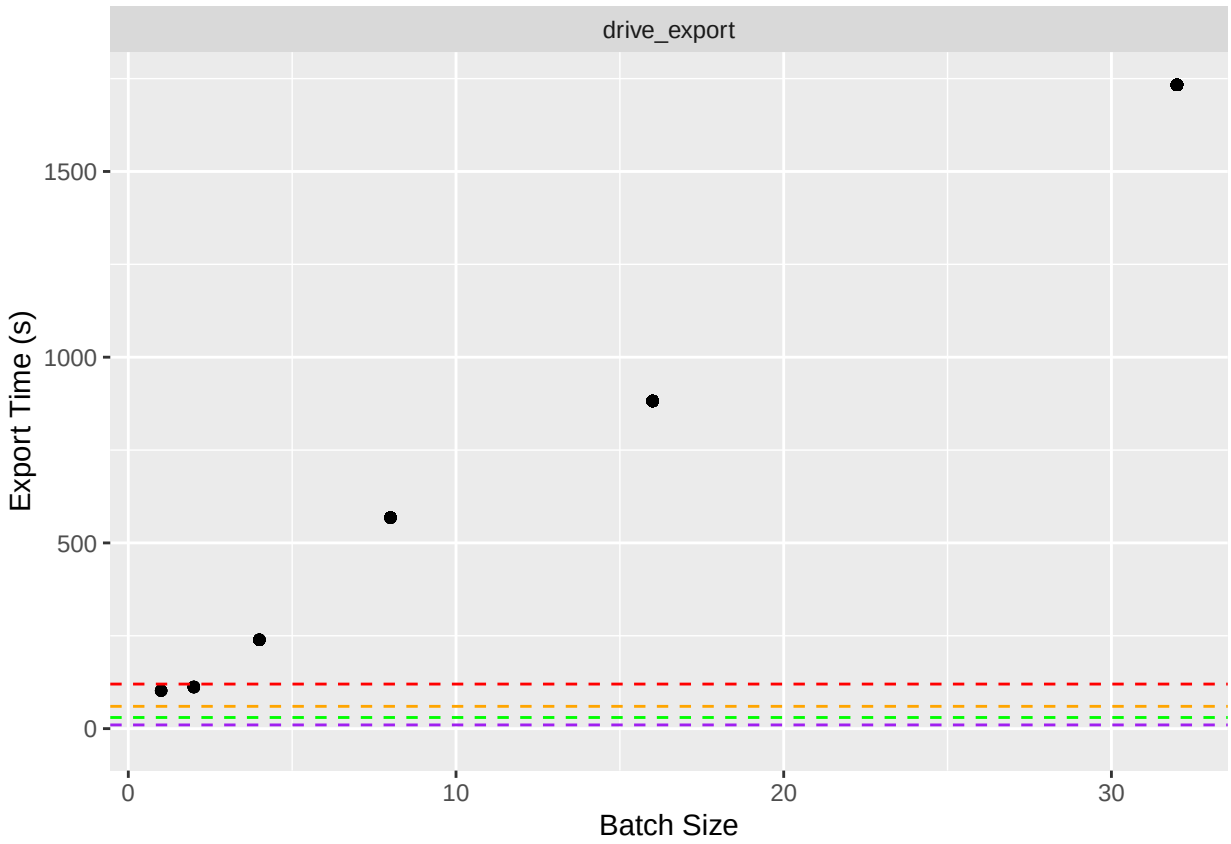
How do job times change as a function of batch-size?



We constructed increasingly large batches of dams (n = 1, 2, 4, 8, 16) consisting of a single small dam replicated n times to explore the effect of batch size on job time.

The plot shows that:

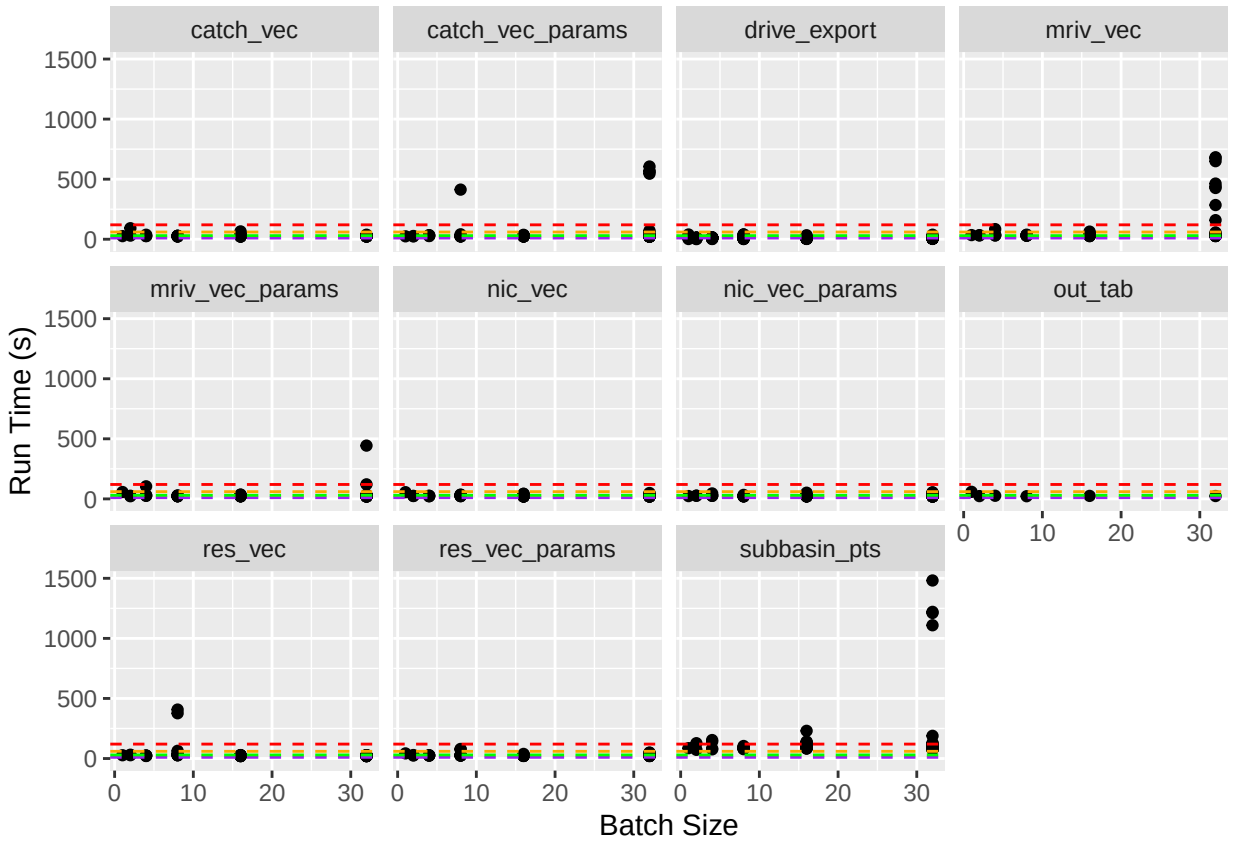
- The average time for processing of a single dam initially falls exponentially and then levels off from 228.80s (batch size = 1) to 759.63s (batch size = 32).
- Despite the early (s-shape) gains in analysis time with batch size, there is a broadly linear increase in total run-time with batch size.



A similar pattern is seen with export time.

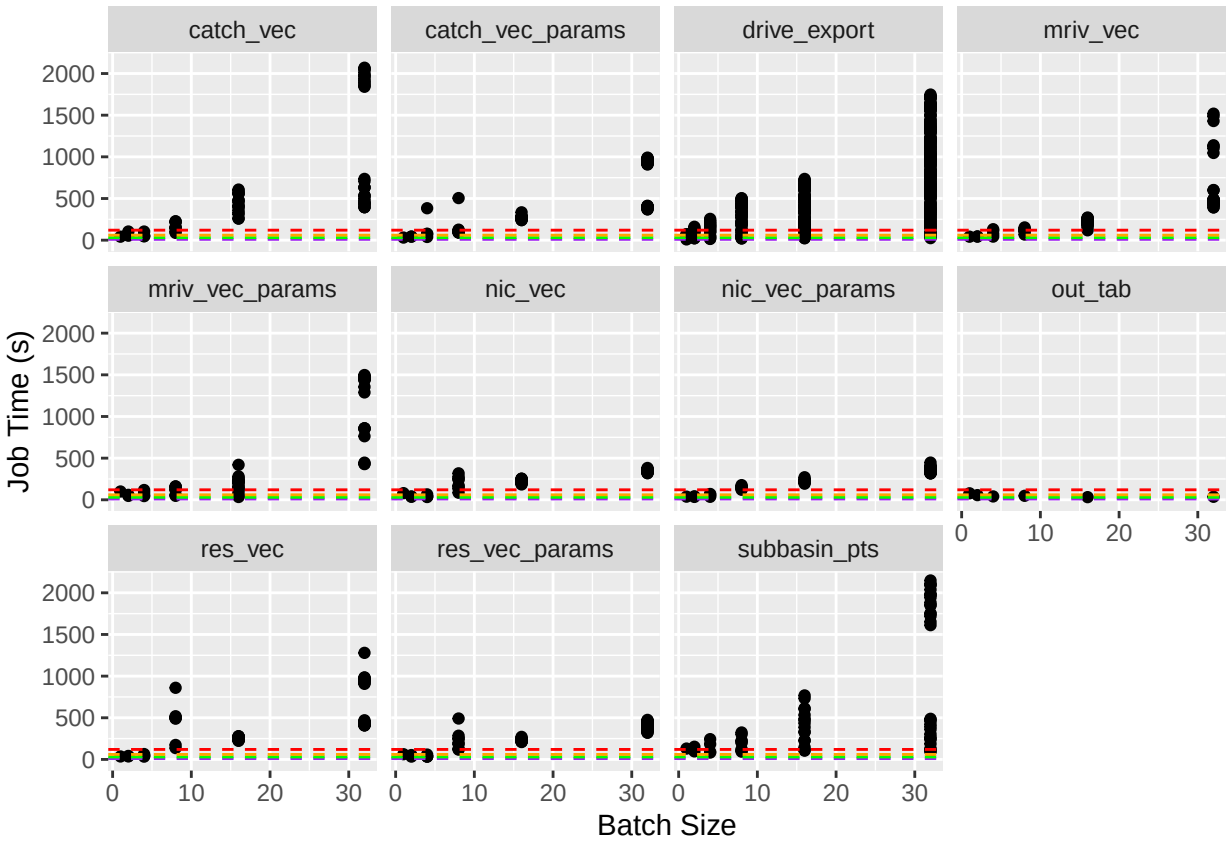
How do task times change as a function of batch-size?

We can look at changes in job time (queue + run time) or run-time (computation only) as a function of batch- size/



We see some odd trends in run-time with batch size:

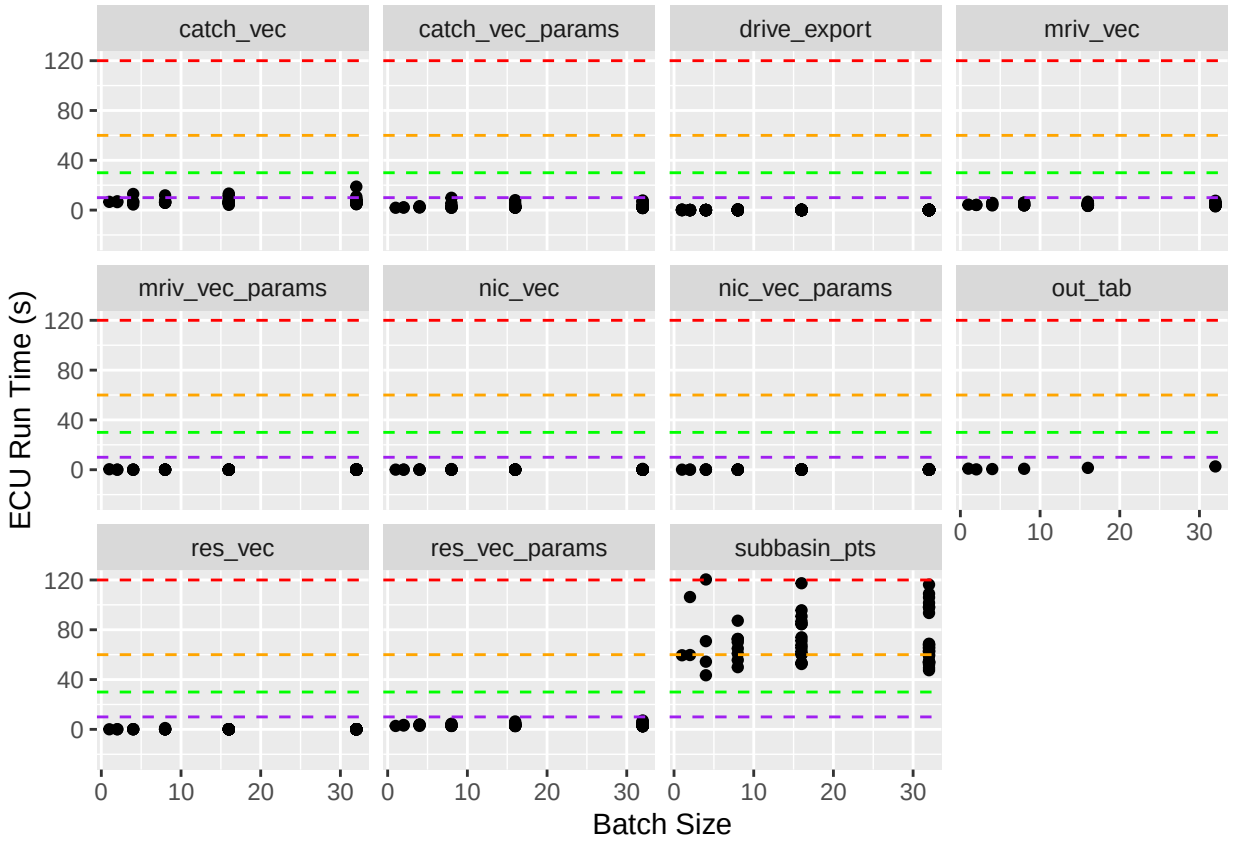
- As batch size increases the spread (variability) of task run-times increases and in some cases becomes bi-modal.



When we take in to account both queue time and run time,

- We see more pronounced increase in job time variability and multimodality with batch size.
- We see a general increase in job time with batch size.

How do batch compute times change as a function of batch-size.



*

Batch compute resource use is fairly consistent across tasks with batch size.

- We see some strange results with subbasin_pt (snapping dam to hydrorivers); resource use and spread of resource use increases remarkably with batch size.

Takeaway messages:

- The time to export to Google Drive increases linearly with batch size to ~30 minutes for 32 dams.
- The Snapping dam locations to hydro-rivers task has the longest run time, the highest batch compute resource use and the highest variability of runtime is a piece of code that should be profiled further.
- Other analysis tasks that should be targeted are:
  - Params main river vector
  - Non-inundated catchment vector
  - Params reservoir vector
  - Main river vector