Metrics

Estimate what's growing from space for $60

Christian Boin + Bdawgkk

March 11, 2022

What the code does

The Pipeline

- Consists of a few dockerized submodules and pachyderm pipeline json files.
- Handles automated ingestion to proprocessing.
  - Fetches them from the USGS API
  - Downloads if they meet certain criteria (low cloud coverage, the right geographical area, etc)
  - Extracts each agricultural quarter section as numerical data

Takes as input the shapefiles of agricultural regions.

Duration per task

```
$ pachctl list job
PIPELINE     DURATION     PROGRESS      DL       UL
ndvi-calc    12 minutes   2  + 0 / 2    747.8MiB 1.304GiB
qs-extract   20 minutes   12 + 0 / 12   1.423GiB 747.8MiB
```

Cost Breakdown

Node	Commitment	Cost per hour
`D8s-v3`	pay-as-you-go	$0.51/h CAD
`D8s-v3`	a 3 year plan	$0.22/h CAD

We will over-estimate and use pay-as-you-go in our calculations.

Stage	Time	Peak Mem Use	Nodes	Cost ($0.51/h)	Parallelism
qs-extract	20m	6G per node	2x D8s v3	$0.34	3/6 bands
ndvi-calc	15m	20GB	1x D8s v3	$0.1275	1 scene

Cost Breakdown (cont.)

Analysis

- Images are not the same size.
- Skew the estimate for one image by taking 75% of the combined cost of two images.
- Esimate for one image: `0.75 * ($0.34 + $0.1275) ~= 0.351`.
- With 120 images a season, **total compute cost of less than** $43**.**

Note: This price accounts for the full cost of pay-as-you-go nodes.

- Shared DAaaS Kubernetes cluster is more likely to make a long-term commitment to a pool of nodes (for a %60 cost reduction)

Storage

For storage cost, hot blob storage costs $0.023/gb/month, so using that as a benchmark, we have, storing two images and the Alberta shapefiles:

```
NAME                    ... SIZE        DESCRIPTION
extracted               ... 1.304GiB    pipeline extracted.
quarter-section-bands   ... 747.8MiB    pipeline quarter-section-bands.
images                  ... 1.374GiB    pipeline images.
events-test             ... 508B
events                  ... 43.98KiB    pipeline events.
timer                   ... 11B         timer for pipeline events.
pathrows                ... 543B        pipeline pathrows.
trackframe-tables       ... 253MiB      pipeline trackframe-tables.
trackframe-shapefiles   ... 1.469GiB
```

Storage (cont.)

With a constant 2gb of storage for the shapefiles and such, plus an estimated 0.75 * (4GB) to estimate the size of an average image & data, we get that 4 months of usage adds up to

```
Constant + Growing
= (2gb * $0.023 * 4 months)
    + ($0.023 / 30) * (3GB + 2*3GB + 3*3GB + ... 120*3GB)
= (2gb * $0.023 * 4 months) + ($0.023 / 30) * 3 * (121*120/2)
= $17
```

This is an upper bound, as we would probably not use hot storage for everything.

Total

The season's estimate would then be around $60 for four months.

What optimizations were done?

Why is it faster and cheaper?

1. We chopped the large shapefile up

Carved up the large shapefile into smaller ones

- Each shapefile roughly the size of a WRS2 TrackFrame.
  - Every satellite image has metadata which states its WRS2 trackframe, so we only need to read in these local shapefiles when processing an image.

Significantly improved the speed and memory footprint of the application.

2. Changed the parallelization code

Originally had several workers processing an image in parallel

- They duplicated the shapefiles and image in memory
- Caused a 30x increase in memory usage.
- Moved to the R `doParallel` package and eliminated this duplication.

3. Avoided writing small files

Originally wrote each quarter section's data to a file

- This is a slow operation
- Instead we grouped the files into a single large file
- We also compressed to minimize storage use.

4. Did Shapefile Preprocessing

Avoid loading the shapefiles

- By extracting the bounding boxes of every quarter section in advance from the shapefiles, we ultimately did not need to load the shapefiles.
  - Reduced the memory footprint by avoiding loading extra metadata.
  - Sped up the computation time by avoiding recalculation on each run.

5. Modified functions from the R raster package

Found a better way to interact with the raster files

- Some of the original approaches to extract one quarter section from the raster image were slow
- Iterated through a few designs and ultimately rewriting a modified and condensed version of the `crop` function from the raster package

6. Ran on Pachyderm & Kubernetes

No warm up or cool down

- Docker images are stored within artifactory, so loading them is very fast.
- The node pool is always live within the Kubernetes cluster, so no warm-up or cool-down time for a vm.

A little parallelization

- quarter section extraction runs on two VMs processing 3 bands and a time.
- The processed images are then passed to one node which calculates the NDVI.

At the moment, this last step has the worst memory performace, but it can easily be fixed, and ultimately it can probably be run with a memory footprint less than 5GB.