Scalable Satellite Imagery Analysis for Automatic Survey of Antarctic Seals using Distributed Computing Frameworks

Nivetha Balasamy [1], Bento Gonçalves[2], Heather Lynch[2], Shantenu Jha[1]
[1] Rutgers University, [2] Stony Brook University

Introduction

In order to assess how changes in sea-ice extent are affecting the Southern Ocean ecosystems, it is pivotal to repeatedly estimate the krill population size. As a group, Antarctic pack-ice seals (Figure 1) are major krill predators and thus seal counts could be used as a proxy for the krill population estimate. Estimating the pack-ice seal population for the entire Antarctic continent is a formidable task. However, since seals are fairly large animals, we can take advantage of high-resolution satellite imagery to count them remotely (Figure 2), avoiding great costs and risks

Figure 1. Antarctic pack-ice seals. Species from left to right, top to bottom: Weddell seal, crabeater seal, leopard seal and Ross seal.

Distributed Computing Frameworks

Detection of seals from satellite imagery of the entire Antarctic continent is computationally challenging and can benefit from parallel implementation of the image processing pipeline. In general Distributed Computing frameworks provide methods and means to handle data and task distribution on cluster nodes. Examples of such frameworks include MapReduce, Storm, Pregel, Dryad, Spark, Scope and Hive etc,.

`Dask.distributed` is a lightweight library for distributed computing in Python. It is a dynamic task scheduler that coordinates several worker processes spread across the cluster. The Dask scheduler is cognizant of the data dependencies of it's workers. It allocates tasks to workers based on this information and computes tasks efficiently by minimizing the data movement between them.

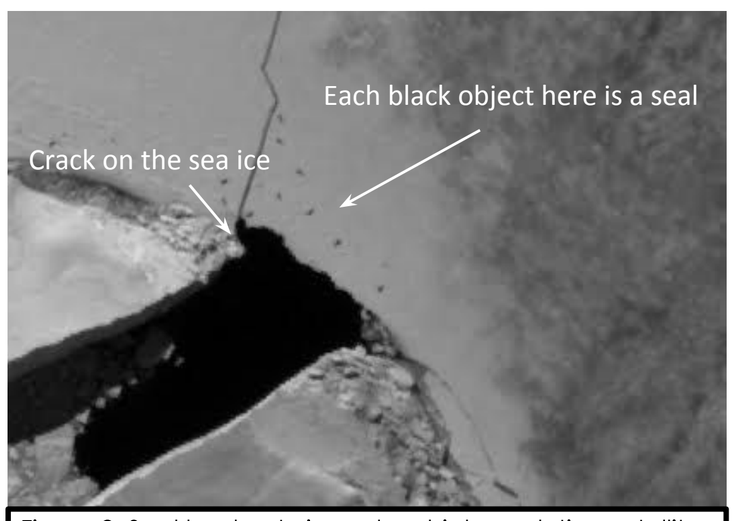
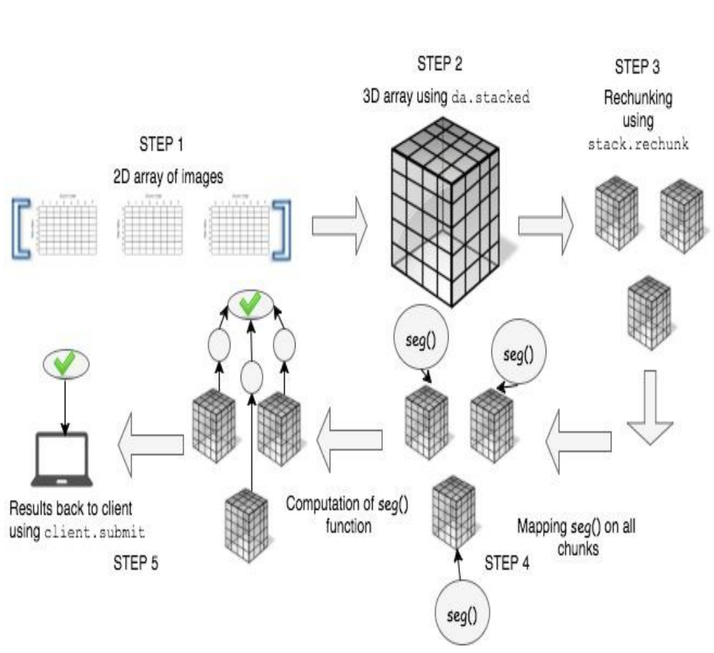


Figure 2. Seal haul out viewed on high-resolution satellite imagery. Haul outs are typically found near cracks on the sea ice.



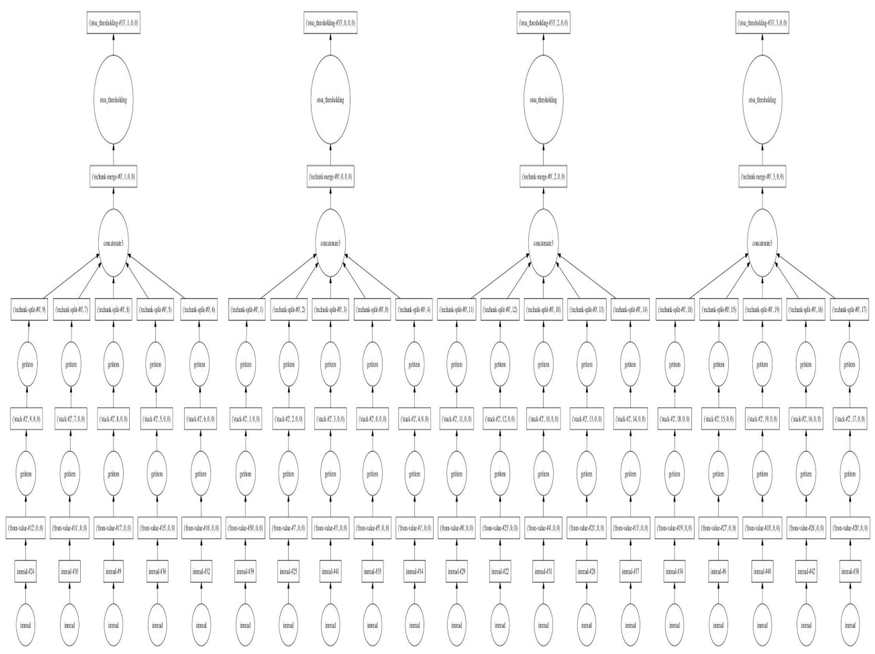


Figure 3. (Top): Pipeline for analyzing images using `Dask.distributed`. (Bottom): `Dask.distributed` task graph for the analysis pipeline

Analyzing Images with `Dask.distributed`

Images can be analyzed in parallel with `numpy` arrays distributed across a cluster using `Dask`. A trivial image processing workflow can be as follows:

- Read images as 2D arrays
- Stitch into a single 3D dask array using `dask.delayed` function.

This gives us a numpy-like abstraction on all the input images

- The entire 3D array is grouped into arrays of dimension 5x2000x2000 (1 chunk) to reduce overhead.

To avoid data transfer delays, data are persisted on the cluster.

- Segmentation operation is mapped on to each chunk
- The results are gathered using the `futures` object back to the client

Object Detection

Since there is a large amount of imagery to be processed, a lightweight CNN, with fewer parameters might be desirable over a heavyweight CNN.

- TF-slim is a new lightweight high-level API of TensorFlow for defining, training and evaluating complex models
- The library supports a rich family of neural network architectures

Dask and TensorFlow

A distributed TensorFlow application is useful when we want to use many concurrent processes to either speed up training or handle large data sets. TensorFlow clusters can be started with Dask using the `dask-tensorflow` library. While TensorFlow remains responsible for all the actual training and scoring, Dask can be used to handle everything else viz., setting up tf workers as long running tasks, gathering results from tf network.

Each TensorFlow (tf) task has a TensorFlow "server" to create and manage sessions and "workers" to actually execute the graph operations. Typically Dask workers can be setup as long running tasks serving TensorFlow computations.

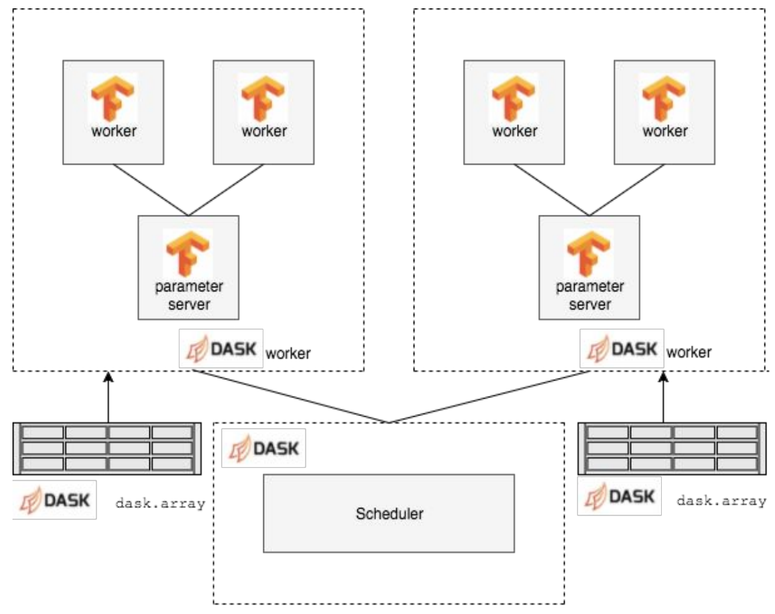


Figure 4. Dask and TensorFlow integration

Preliminary Results

A mean of all images was computed with this approach. For a sample size of 20 images (4 chunks), the mean pixel values before and after thresholding are 12.62 and 0.10 respectively. This is achieved in order of milliseconds.

Next Steps

- Characterize Dask's performance using larger datasets for strong and weak scaling
- Understand communication costs and load balancing among various workers

References

1. Matthew Rocklin, Dask: Parallel Computation with Blocked algorithms and Task Scheduling, Proc. of the 14th Python in Science Conference, 2015
2. R. Cresson and G. Hautreux, "A Generic Framework for the Development of Geospatial Processing Pipelines on Clusters," in *IEEE Geoscience and Remote Sensing,* Nov. 2016.