Image Processing Using MapReduce and Hadoop

Alexandru Ionascu, William Daniel Laszlo

Distributed Systems Lab Report, January 2019

1 Introduction

In this report we will present two common application in image processing: grayscale conversion and blurring. The challenge of this task consist in the size of the input. We will implement a MapReduce algorithm for each task and we will run it on Google's Cloud Dataproc service.

In order to approach testing in the proposed algorithms in an easier way, we will use our custom format for generating matrices of pixels. Our input will be loaded into Google File System which is a proprietary distributed file system developed by Google to provide efficient, reliable access to data using large clusters of commodity hardware. We will try to run in different scenarios to see how can we improve our algorithm by the number of parallel tasks.

2 Grayscale Conversion

This task is straightforward, and can be done similar to a "Hello "World" program in MapReduce. The only relevant process is the mapper. Our input image consists in a matrix of $N \times M \times 3$, where the values are between 0-255. We will compute the rec601 luma (Y') component computed as:

$Y' = 0.299R + 0.587G + 0.114B$

The input is being split by the newline character and each mapper will process an individual row of the image at a time. The mapper job implementation in Python 3 is shown below.

```
def mapper(self, _, line):
        rgbs = [int(x) for x in line.split(' ')]

        for i in range(2, len(rgbs), 3):
            y = int(
                0.299 * rgbs[i - 2] +
                0.587 * rgbs[i - 1] +
                0.114 * rgbs[i]
            )
            yield _, y
```

3 Image Blurring

This is done by convolving image with a normalized box filter. It simply takes the average of all the pixels under kernel area and replace the central element.

Figure 1: Comparison between the original image and the blurred one with the kernel defined below, side by side [1]

The kernel is defined as $Ker = \begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}$

Similarly, we split the image by new line and the mappers will process them one by one, but this time we take into account the neighbors of the pixel. The blurred color of the pixel is the average of it's 8 neighbors and the pixel itself. The mappers will produce 9 times more key-value pairs than the actual input.

```
def mapper(self, key, line):
        x = line.split(' ')
        row_ind = int(x[0])
        n = len(x)
        directions = [
            (-1, -1), (-1, 1), (1, -1),
            (-1, 0),  (0, -1), (0, 0),
            (1, 0),   (0, 1),  (1, 1)
        ]

        for i in range(1, len(x)):
            for dir in directions:
                row = row_ind + dir[0]
                col = i + dir[1]

                if row >= 1 and col >= 1:
                    yield '{0},{1}'.format(row, col), int(x[i])
```

Reducers will take the pixel colors and will perform the sum and then the multiplication with $\frac{1}{9}$.

```
def reducer(self, coords, colors):
    yield coords, int(sum(colors) * (1/9))
```

4 Benchmark

We will use two different generated test files, a smaller one measuring 89.3 MB with 5000 x 5000 pixels and a larger one measuring 200.9 MB with 7500 x 7500 pixels.

We will measure the performance for blurring, since it's the most demanding. We will choose our instances to be n1-high-cpu-4, since n1-high-cpu-2 doesn't have enough memory, and n1-high-cpu-8 exceeds our quota.

For each number of instances, Dataproc automatically launches a predefined number of mappers and reducers as shown in the metrics below.

- 89,3 MB image (5000 x 5000) –instance-type n1-high-cpu-4 –num-core-instances-4: 40 mappers, 17 reducers: 14 min 40 sec
- 89,3 MB image (5000 x 5000) –instance-type n1-high-cpu-4 –num-core-instances-3: 30 mappers, 13 reducers: 17 min 16 sec
- 89,3 MB image (5000 x 5000) –instance-type n1-high-cpu-4 –num-core-instances-2: 20 mappers, 8 reducers: 24 min 29 sec

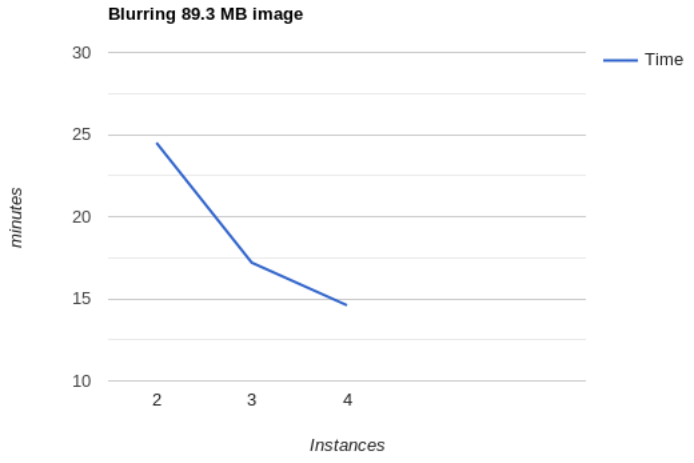

Figure 2: Performance for blurring a smaller image with n1-high-cpu-4 instances

- 200,9 MB image (7500 x 7500) –instance-type n1-high-cpu-4 –num-core-instances-5: 50 mappers, 21 reducers: 22 min 49 sec
- 200,9 MB image (7500 x 7500) –instance-type n1-high-cpu-4 –num-core-instances-4: 40 mappers, 17 reducers: 24 min 55 sec
- 200,9 MB image (7500 x 7500) –instance-type n1-high-cpu-4 –num-core-instances-3: 30 mappers, 13 reducers: 32 min 53 sec
- 200,9 MB image (7500 x 7500) –instance-type n1-high-cpu-4 –num-core-instances-2: 20 mappers, 8 reducers: 48 min 42 sec

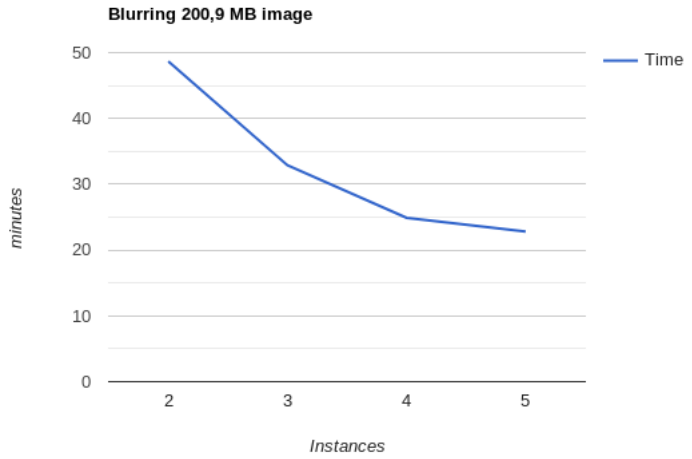

Figure 3: Performance for blurring a larger image with n1-high-cpu-4 instances

References

[1] Victor Powell. *Image Kernels.*