CS 4480 Data-Intensive Computing

Project Group 5

Mosaic images generated with Map-reduce and Spark technique

Final Report

YAU Wing Ho Kennedy (5357 3320)
TSE Kwok Chiu Shayne (5355 3634)
Paroma Chatterjee (5415 1354)
NG Tsz Ho Dickson (5360 9486)

Table of Contents

1. Introduction

1.1 Background and problem statement

Originally, a mosaic is a piece of art or image made from the assemblage of small pieces of colored glass, stone, or other materials.Gradually, thanks to the advance of image processing technique and computing power, mosaic could be easily digitally generated.

In the project, the idea of the mosaic is a primary image, also called target image (High-resolution image) that divided into different tiled sections which the tiled sections replaced by other small images (the dataset size of low pixel image is around 250GB). When looking closely at the mosaic, small images could be observed; when looking at the mosaic a bit further, the primary image becomes the dominant and the most obvious image.

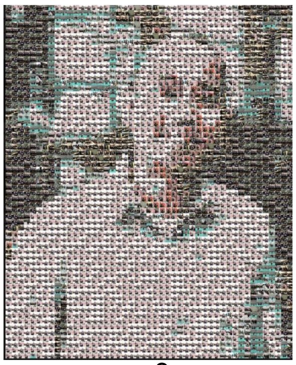
When it comes to enormous dataset, computing power and data transfer always become the main concerns. Therefore, we choose to implement our solutions with data-intensive technique shown in later parts.

1.2 Objective

The aims of the project is to investigate the potential improvement for the Brute-Force solution by data-intensive approaches, so as to understand better the role of data-intensive computing in image processing field.

1.3 Motivation

Reasons for selecting this topic are in three-folds: Meaningful image fusion, New trend for images management, Image processing with Data-Intensive computing technique



1.3.1 Meaningful image fusion

First of all, a mosaic is a form of art which gathers different small images to generate a given target image and that could become a meaningful gift- image/memory fusion connecting both sender and receiver. The mosaic image is an example for the case, which consists of many graduation photos of a CityU student and the target image is the professor who taught him before. To our belief, nothing could be more heart-warming and meaningful than such a technical image fusion gift for Computer Science professor.

1.3.2 New trend for images management

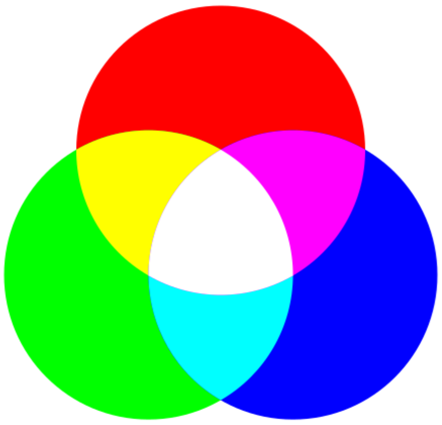
Besides, more and more people like to take a photo by their mobile gadget such as smartphone or camera. Mosaic could be either a sort of filtering or even a way for better storage management- to store innumerable photos into a single one. If making an apps would be the next step, then we surely need more efficient methods for handling tons of images in short period of time, if not in an instant.

1.3.3 Image processing with Data-Intensive computing technique

In the 21st century, visual data/image processing has become even more persuasive and popular, but obviously, one of the bottlenecks is the processing speed. Therefore, throughout the project, we expect not only to deepen our understanding of Data-Intensive computing, but to sharpen our skills in image processing at the same time.

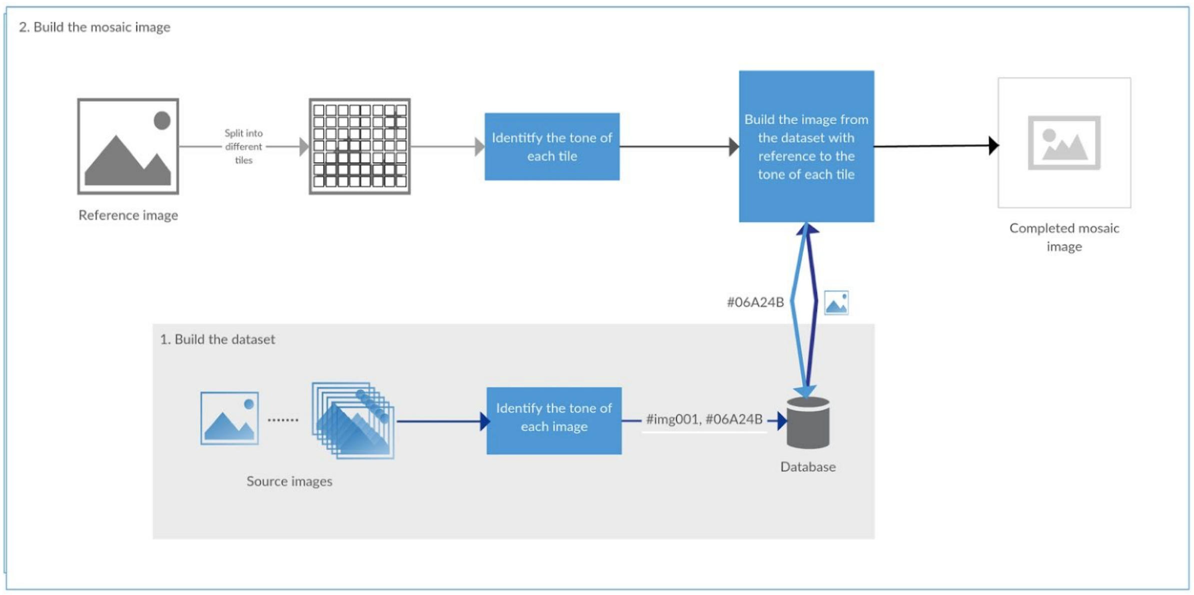
2. Basic solution(baseline)

2.1 Idea



The basic solution of our project is BRUTE-FORCE FOR LOOP with simply for-loop to compare each small image with each tile of the target image. The matching method of basic solution is to calculate the color distance in each dimension of RGB (graph) of each small images and tile in target image. The solution is relatively simple and widely used, so we decide to use it as baseline for comparison and performance evaluation in the following parts.

2.2 Mechanism/Design


2. Build the mosaic image
Reference image
Split into different tiles
Identify the tone of each tile
Build the image from the dataset with reference to the tone of each tile
Completed mosaic image
#06A24B
1. Build the dataset
Source images
Identify the tone of each image
#img001, #06A24B
Database

The basic solution consists of two stage, building the dataset and creating the mosaic image.

For the first stage, building the dataset, all smaller images that used to composite the mosaic image are located in the same directory initially. Then, we will loop through the directory and identify each image's tone one-by-one. The image's file name and the tone color would be stored in the database after processing.

In the second stage, building the mosaic image, firstly, we have to input 1 image that as the reference image for the mosaic image. Then, we would split the image as different tiles based on settings. Each tile would then calculating its’ respective tone color. The tone color of each tile would be used to compare with the database’s data to find entry that has the smallest Euclidean distance. After all tile has found the respective smaller images, the small images would then be used as resources to render the mosaic image.

2.3 Problem of the Solution

Computational Time and Cost is high
The dataset of small image is larger and the target image is a high pixel. The basic solution is compared each small image with each tile section, the number of the small image around 80,000,000 for our project and the target images is divided many tile sections according to the pixel. An operation of each tile section is 80,000,000 times and the whole operation of a mosaic image are more than 100,000,000 usually.

Therefore, the running time of mosaic processing will be very high, it may be more than an hour. Since the time is very long and the solution is lengthy, we haven’t run all dataset in basic solution.

2.4 Short summary

The efficiency of the basic solution, this solution is a redundant and clumsy solution. Since the process time of a high-resolution image with dataset need more than an hour, when we have hunger high-resolution image, the time requirement maybe need more day. This is inefficiency solution for process larger dataset. Therefore, we use another method to handle the larger image set. The k-means and map-reduce are the advance solution. We will compare the time requirement with different solutions and find the most efficient method to handle the larger dataset.

3. Color-Bucket Solution (Spark Mapreduce)

3.1 Motivation

The main problem with the brute-force solution is that for every tile in the big image, all the small images in the dataset need to be compared in order to find the most accurate image for that tile. We seek to parallelize the comparisons using a mapreduce approach, and make use of the reducer property of aggregating values for a given key.

3.2 Mechanism & design

This solution is a way of using partitioning by key to reduce the number of comparisons that have to be made in order to find a suitable image for each tile. The algorithm operates in two map-reduce operations : first for the target image, and second, for the big dataset of small images from which we wish to construct the target image.

In the first operation, we split the big image into tiles and calculate their averages (map). We then group these tiles by tile color into what we refer to as buckets (reduce). Buckets are associative arrays with the key being the tile color and the value being the list of coordinates of tiles that have that color.

The result of the above operation is collected as a map, and sent as a broadcast variable to the next operation, in order to facilitate matching.

In the second operation, we calculate the image average of each small image and map it to the closest key (color of a tile in the big image) in the bucket broadcast variable.

The reduce part of this phase takes a bucket color as key, and finds the small image with the closest average to itself. Thereby, the number of values that need to be compared for each tile in the big image is greatly reduced, and the process is more parallelised.

3.2.1 Map reduce algorithm to find the correct small image for a tile in the big image

```
class TARGET_IMAGE_MAPPER
    method MAP (key  : filename, val : filebytes)
        tiles[] ← SPLIT_INTO_TILES(filebytes)
        for all tile t ∈ tiles do
            tileCoordinates, tileRgb ← t
            colorAvg ← CALCULATE_AVG_color(tile_rgb)
            bucketcolor ← colorAvg
            EMIT (bucketcolor , tileCoordinates)

class TARGET_IMAGE_REDUCER
    method REDUCE (key : bucketcolor, val : coordlist [c1, c2, ... ])
        EMIT (bucketcolor, coordlist[c1, c2, …])


/* collect reducer output from stage 1 as a HashMap / dictionary  of (color_bucket,
coord_list[…]) pairs
  store in a variable called bucketMap
 buckets ← broadcast(bucketMap) so it can be sent as is to each reducer as a constant */


class SMALL_IMG_MAPPER
    const CONST_MAX ← 277
    buckets ← GET_VALUE_OF_BROADCAST(buckets)
    method MAP (key : filename, val : filebytes)
        smallimgAvg ← CALCULATE_AVG (val)
        minDistance ← CONST_MAX
        bestBucketcolor ← GET_RANDOM_BUCKET(buckets)
        for all (bucketcolor, bucketCoordinates) ∈  buckets do
            distance ← CALC_EUCLIDEAN_DISTANCE(bucketcolor, filename)
            if distance < minDistance do
                bestBucket ← bucket
                minDistance ← distance
        EMIT (bestBucketcolor, [minDistance, filename])


class SMALL_IMG _REDUCER
    method REDUCE (key : bucketcolor, val : pairs [(minDistance1, filename1),
[(minDistance2, filename2),... ])
        filename, minDistance ← MIN_DISTANCE(pairs)
        EMIT (bucketcolor, filename)
```

3.2.2 Euclidean distance

The Euclidean distance between two RGB vectors [r1 b1 g1] and [r2 b2 g2] is given by

$$D = \sqrt{[(r1 - r2) ^2 + (b1 - b2) ^2 + (g1 - g2) ^2]}$$

The advantage of this metric is that it is simple to calculate, however there are colour schemes that more accurately replicate the human perception of colour distance such as the CIE L*a*b colour space.

3.3 Advantage

The major advantage of this solution over the naïve “for-loop” solution in part 2, is that there is a degree of parallelism in finding the most suitable small image for a particular tile in the big image.

The first disadvantage is that the reduce by key process leads to lots of shuffling in stage 3 and is the biggest contributor to total time (please see screenshots).

3.4 Limitation

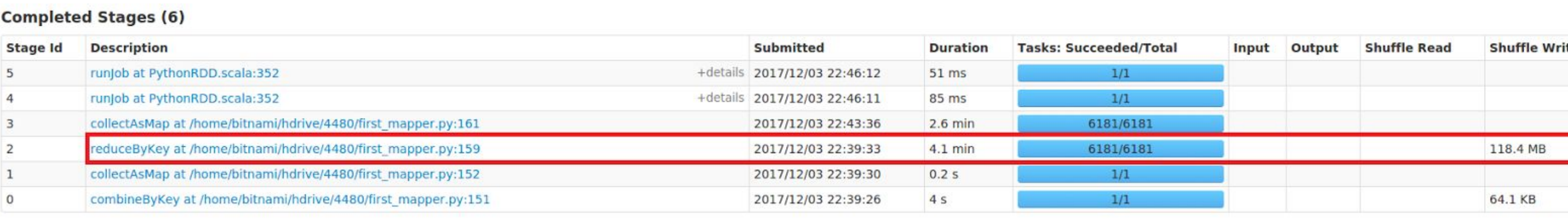
We also observed that reading from the HDFS is still slow, so if we want to reconstruct the target image by reading the small images one by one from HDFS, then it takes a very long time. It would be better if, during the reconstruction stage, the small images could be read from local disk.

3.5 Performance

If more partitions are specified while reading all the image as sequence file, the better is the parallelism and the stage runs much faster.

The memory used by the broadcast variable was 25 KB for tile size 8px by 8px.

The bottleneck is at the reduceByKey part, where the key value pairs generated by the SMALL_IMAGE_MAPPER are shuffled into their respective reducers. The image is showing the case for a run with 8x8 tiles and 9450 small images.



Completed Stages (6)

Stage Id	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Wri
5	runJob at PythonRDD.scala:352	+details	2017/12/03 22:46:12	51 ms	1/1				
4	runJob at PythonRDD.scala:352	+details	2017/12/03 22:46:11	85 ms	1/1				
3	collectAsMap at /home/bitnami/hdrive/4480/first_mapper.py:161		2017/12/03 22:43:36	2.6 min	6181/6181				
2	reduceByKey at /home/bitnami/hdrive/4480/first_mapper.py:159		2017/12/03 22:39:33	4.1 min	6181/6181				118.4 MB
1	collectAsMap at /home/bitnami/hdrive/4480/first_mapper.py:152		2017/12/03 22:39:30	0.2 s	1/1				
0	combineByKey at /home/bitnami/hdrive/4480/first_mapper.py:151		2017/12/03 22:39:26	4 s	1/1				64.1 KB

Time taken for stages of application (not including the time to reconstruct image)

Stage 0 and 1 are the jobs for the computations for the large image. Stage 2 is for computation for small images (reduceByKey to group the small images into buckets) In stage 3, we collect output from stage 2 as map for image reconstruction. The results of one test are given below.

Fixed input size : 5000 images

Fixed target image size: 768 x 1024 px

We varied the number of splits in reading the small images in the binaryFile format and obtained the following results:

	no splits	8 splits
16x16 tiles	Stage 0 : 3s Stage 1: 0.1s Stage 2: 1.5min Stage 3: 42s Including time to put together images : 5.5min	Stage 0 : 3s Stage 1: 0.1s Stage 2: 1.3m Stage 3: 42s Including time to put together images : 5.5min
8x8 tiles	Stage 0 : 5s Stage 1: 0.2s Stage 2: 1.7min Stage 3: 42s Including time to put together images : 13.6min	Stage 0 : 5s Stage 1: 0.2s Stage 2: 2.0m Stage 3: 43s Including time to put together images : 9.6min

The image takes a long time to put together because, once the correct images have been identified for each tile, those images need to be read one by one from HDFS again (as per our current implementation). During comparisons with the other solutions we decided to disregard this last part in our time calculations, as the pictures can be read from local disk and that is much faster as demonstrated by the recorded time for different stages of the next solution

We can see that the biggest bottleneck is the reduceByKey part (stage 2), in which the images are shuffled into their respective buckets. The collectAsMap() phase, which is stage 3, takes constant time for all cases.

4. K-Means Clustering Solution

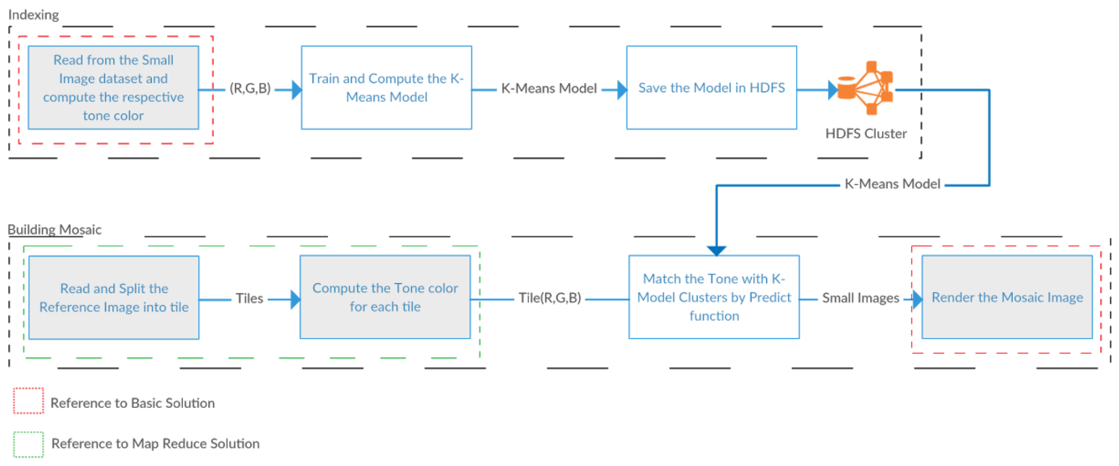
4.1. Motivation

After we have implemented the solution based on Map-Reduce, we found that although the solution is able to meet the initial requirement for the project, the performance has not achieved the expected requirement. One of the main reason for the former solution could not further improve the performance is that it requires to loop through and retrieve the images to calculate the color distance between the image set and each tile which trigger a relatively high computation and I/O cost.

In order to reduce the cost of calculating the color distance, having a better pre-processing and clustering on the image dataset could further improve the solution. K-Means clustering has been chosen as the method for clustering the dataset. Compare to other clustering solution, i.e. clusters based on color families, K-Means clustering could be more adapted to different dataset composition's characteristic and shown in clustering outcome.

4.2 Mechanism and design

The improved solution is based on the Map-Reduce solution and the original basic solution. We have added the modified indexing stage from the basic solution to the Map-Reduce solution. The overall design of the improved solution is shown below:


Indexing
Read from the Small Image dataset and compute the respective tone color
(R,G,B)
Train and Compute the K-Means Model
K-Means Model
Save the Model in HDFS
HDFS Cluster
K-Means Model
Building Mosaic
Read and Split the Reference Image into tile
Tiles
Compute the Tone color for each tile
Tile(R,G,B)
Match the Tone with K-Model Clusters by Predict function
Small Images
Render the Mosaic Image
Reference to Basic Solution
Reference to Map Reduce Solution

For the K-Means model training, we have put the R, G, and B values from each small image in the dataset as three-dimensional vectors for training. After the model has been trained, it would then be stored in the HDFS. Then, during the stage for creating the mosaic image, the model will be firstly loaded from the HDFS with the target image. Soon after both resources have been loaded into the memory, it will then follow the former solution to calculate each tile's tone. Hence, the tone in R, G, and B value would be used by the K-Means model to predict the target cluster of the suitable smaller images that suit the tile's tone respectively. Finally, it would follow the basic solution to render the mosaic image using the smaller images.

Pseudo code

```
FUNCTION Indexer:
    FOREACH image FROM dataset:
        ToneRGBs.append(getTone(image));
    Model = KMeansModel.train(ToneRGBs, NumberOfClusters);
    Model.save(sc,"kmModel");

FUNCTION MosaicProduce:
    /** Above same as the Map-Reduce Solution**/
    Model = KMeansModel.load("kmModel");
    FOREACH tile FROM tiles:
        ImageList.append(Model.predict(tile.getTone));
    /** Below same as the Basic Solution to produce the Mosaic Image **/
```

4.3 Advantage

The improved solution provides the following advantages:

4.3.1 Faster speed for rendering more than 1 mosaic image

One of the advantages of using the K-Means model is that after the model has been trained in the indexing stage. The same model can be repeatedly used as long as the dataset has not changed. As the process time for using a model to predicting a cluster is close to negligible, the matching time during building the mosaic image can be significantly reduced.

4.3.2 Dynamic clustering

The K-Means model cluster the dataset purely based on the vectors provided. Hence, the clusters produced by K-Means clustering is dynamic that matches the dataset actual characteristics. Hence, in each cluster, it would have a relatively equal number of data (smaller image). For example, for a dataset contains lots of blue images, the K-Means clustering method can precisely separate different degree of blue tone images which in another form of clustering method that may not able to do unless amending the implementations.

4.3.3 Utilization of in-memory processing

The K-Means model is computed and stored as an RDD object that would be loaded and processed in memory. Hence, most of the I/O processes has reduced as the only item need to be load from the HDFS is model itself after the model has been created.

4.4 Limitation

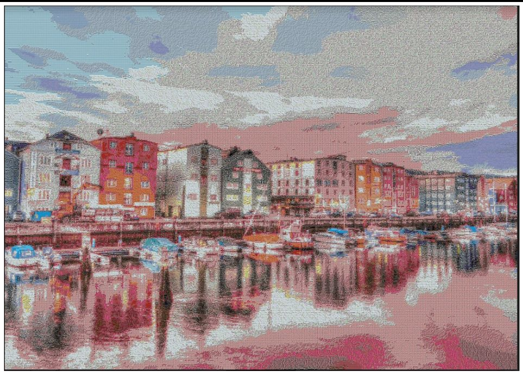
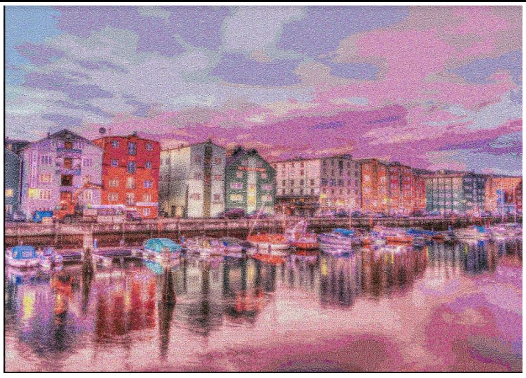
The improved solution faces the following limitation:

4.4.1 Relatively longer time for training the K-Means Model

When the amount the data (smaller images) grows, it would take a significantly longer time to train the K-Means model when compared with the basic solution's indexing process time. Besides, the read and write time for the K-Means model from HDFS are non-negligible

4.4.2 Number of clusters in K-Means Model training affects the quality of the Mosaic

When the number of clusters in the K-Means model training increase, it would significantly increase the processing time for training the model. However, with more clusters, the quality of the Mosaic would improve as more accurately match the tile's tone color. We have to find a balance between the processing time and the quality of the Mosaic.

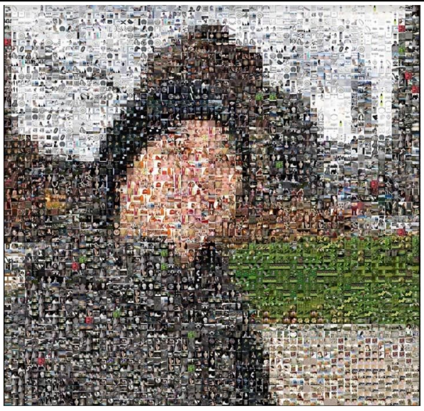
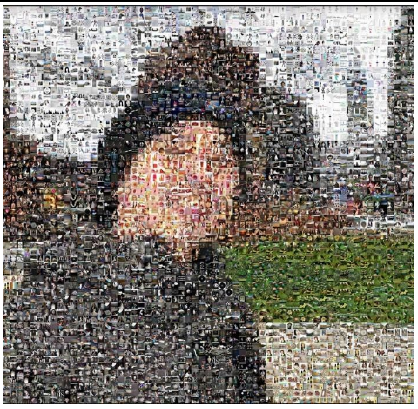
	
Cluster Setting: K=256	Cluster Setting: K=2048

In the above table, we can see that the number of clusters can significantly affect the quality of mosaic as the number of clusters actually equals to numbers of colors. During series of testing, we found that K=1024 is one of the suitable numbers of clusters that balance the quality and processing time.

4.5 Performance

The improved solution with K-Means Clustering has significantly improved the performance especially when the number of tiles and number of smaller images growth. A list of experimental result compare against with the basic solution is listed below:

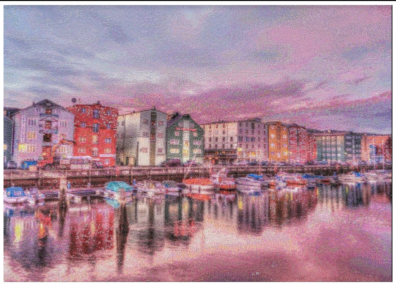
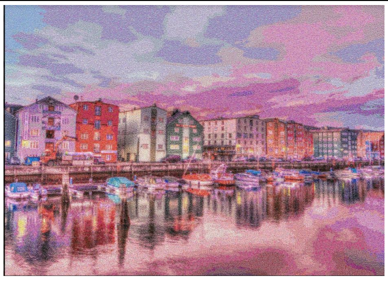
Test: #1

Input Image	**Basic Solution**	**Improved Solution with K-Means**
		

Tile Size: 100px, No. of Tile: 3969, No. of Smaller Images: 5000

Process	**Basic Solution**	**Improved Solution with K-Means** (K=1024)
Indexing	13s	42s
Building Mosaic	130s	43s
Total Time:	143s	85s

Test: #2

Input Image	**Basic Solution**	**Improved Solution with K-Means**
		

Tile Size: 15px, No. of Tile: 3969, No. of Smaller Images: 20000

Process	**Basic Solution**	**Improved Solution with K-Means** (K=2048)
Indexing	51s	90s
Building Mosaic	4948s	371s
Total Time:	4999s	461s

From the above testing result, we could conclude that the solution with K-Means clustering can significantly speed up the processing time. When comparing the product quality, there are only minor differences between the basic solution and the improved solution. If the data size increase to a certain level, the process time with the improved solution can be just 1/10 of the original solution. It proves that the improved solution can speed up the process without sacrificing the quality.

Testing data and Result

Tile Size	**No. of Smaller Images**	**No. of Tile**	**Indexing Time (K=1024)**	**Building Mosaic**
50px	21000	15481s	93s	90s
60px	21000	9523s	93s	74s
50px	5000	9047s	42s	62s

4.6 Short summary

By implementing K-Means clustering for improving the solution, the process time is significantly reduced especially when the numbers of smaller images and tiles grow. The process time increase due to the training of K-Means model is completely canceled out by the benefit brought by cluster prediction during the matching stage.

5. Solutions comparison and Evaluation

Case	No. of small images	Tile Size	Number of Tiles	Brute-Force For Loop (Baseline)	MapReduce (not including time to reconstruct) (Solution 1)	K-means (Solution2)
1	2000	16px	3072	~89s	100s	22+13
2	5000	8px	12288	~330s	160s	42+34
3	20000	4px	49152	~1500s	300s	**90+119**

The table above shows the comparison in different cases between different solutions: Brute-Force For Loop(Baseline), MapReduce(Solution 1), K-means(Solution 2).

In any of the above cases, result from Solution 1 & 2 transcends the baseline solution, with the most significant time difference in case 3. The higher the number of tiles could make a much significant difference between solution 2 with solution 1 and baseline solution.

For Solution 1, the figure does not include the time for reconstructing the target image, because the time for I/O from HDFS clusters takes too unreasonably long and the problem has been raised in the previous parts.

For Solution 2, after considering long I/O time and matching workflow, the solution takes around 1/10 time of the baseline solution and less than 1/6 time of the full Solution 1. The small images are read just once from the drive and only those images(equals to number of tiles in target image) for constructing the targeted image would be read again, which is not significant. Another part that worth mentioning is the number of cluster in the cases for Solution 2. The Solution 2 has used 1024 clusters per cases, which could already generate quite high quality mosaic image (could be found in chapter 4) that is comparable to or even better than the Brute-Force For Loop Solution.

6. Reflection & discussion

6.2 Trade-off

The following are the main trade-off and struggle points along with project:

6.2.1 MapReduce effect vs compile time

The size of each tile in target image implies the number of tile. The smaller tile size is set, the more tiles there are. In the Solution 1 (MapReduce), when the number of tiles increases, the feature of parallelism could be utilized more efficiently for matching the tiles in target images with small images within the buckets.

However, the reduce process is not guaranteed to be so helpful due to the uncertainty of tiles' color distribution, which may require even more time for more number of tiles to be compiled.

6.2.2 Quality vs time

In Solution 2 (K-means), one core factor determining both processing time and image quality is number of cluster. Generally speaking, the worst-case time complexity of K-means would be considered as O(nkti). Although the correlation between number of iteration and number of centre are still unclear, by our testing result, the number of cluster has great impact to the processing time.

At the same time, the more clusters there are, the higher quality the image would be, because the number of clusters means the total number of different colors that could be identified in the target image.

The assumption of the trade-off above is that more diverse color is equal to better quality of the generated Mosaic. Nonetheless, the fact observed from our project is that once the number of clusters has reached certain level (e.g 1024), the difference is not significant to human eyes. Suggestively, it could also be considered as a filtered or stylish mosaic, because the art of Mosaics is not restricted to the high replication of the image but its aesthetic meaning as well.

6.2.3 Fault tolerance vs I/O time

Although the fault tolerance may not be the focus in this project, because of the comparatively short running duration (e.g with 1 day) and the lower impotence of the integrity the small images or pixels. By experimental purpose, the Hadoop clustering has been adopted in the solution 1 (MapReduce) and the I/O time of it is proved to be too significantly unsatisfactory in the project.

In real environment, it is suggested to consider more carefully whether to use the HDFS clustering for each individual scenario for better fault tolerance in exchange of more I/O time.

6.3 Bottleneck

6.3.1 I/O time

I/O time is significant especially when data size is not large enough (e.g several TBs) with not so large data processing time at the same time- the problem could be even worsening for HDFS I/O. However, the problem is almost unavoidable for image processing problem, because the pixel of each images in target image and small images are expected to be read at least once.

Thus, part of our solution's aims is to minimize the I/O time, for example, in Solution 2, after K-means models trained, the system would not read the small images again until the best match cluster has been found through the trained K-means model. Nonetheless, the time for training the model requires at least reading the small images for once, which means the I/O time is still inevitable.

6.3.2 Image manipulation time

As image consists of pixels, in order to get meaningful information with respect to "color", the manipulation (e.g average) is unavoidable. For all the solutions in the project, one of the most time-consuming processes is to calculate the average number in each dimension of RGB among the pixels in small images, which is hard to be improved by any data-intensive technique due to feature of "each image as a unit". The problem could be worsening when the manipulation becomes any more complicated. Therefore, the performance of image manipulation could be a bottleneck.

7. Conclusion

After evaluation and comparison, the best solution is Solution 2 (K-means). The main reasons for that are in three-folds: 1. reduction for I/O time 2. Running with RDD 3. Clustering.

Thorough the whole project, there are much more consideration than what we expected, for example, the technique required for image processing, the runtime for HDFS, the workflow of the solutions, the unavoidable I/O time and so on. Every problem has driven us to brainstorm more potential ways, not limited to data-intensive computing ways, to improve the solution. One undeniable fact is that the improvement by data-intensive computing technique is significant, especially when the dataset is large enough.

Last but not least, it is believed that every solution is improved bit by bit with joint effort in different areas, like algorithm, image manipulation method, the sense of data and etc, and data-intensive computing is just one element of it, specifically when the project scaling up. The project has deepened our understanding in big-data related project in a variety of fields. It is suggested to encourage the fellow students to attempt on more creative and interesting projects in the future.

8. Future development

8.1 Clustering algorithm

For Solution2, the clustering algorithm is one of the key factors regarding to the time for massive dataset training. Regarding to deployment time, k-means is reasonably fast for clustering the new incoming data, which has been shown in previous part. However, there is still a room for training time. Deployment time and training could also be one of the trade-off in the algorithm level. Therefore, more testing and experiment could be done for different clustering algorithm (e.g Approximate Nearest Neighbour (ANN) and Locality Sensitive hashing (LSH))- in the examples, the two algorithms don't directly solve the clustering problem, but they would be able to tell which points are "close" to one another.

8.2 Way of image manipulation

In 3 of the solutions in the project, the color is used as the core factor for matching the tile with small images but the way of image manipulation could BE a lot, e.g the edges detection and matching with more advanced machine learning techniques. The result is that the matching could be more suitable after considering the edges as well and the quality of the generated Mosaics image could be higher even with small number of tiles.

9. Individual Contribution statement

9.1 Yau Wing Ho Kennedy

Contribution:

1. Investigation and implementation for image processing
2. Brainstorming and making up solution 2 with Shayne
3. Presentation slides
4. Solution Comparison and Evaluation, Reflection and Discussion and Future Development, alignment and formatting in final report

Reflection:

The project is quite challenging for most of us, in terms of the understanding of the course, the mindset of efficient data computing, image processing technique, python programming and so on, but I am sure that all of us have been inspired and pushed forward in the field by the project after countless discussion and strugglings among us.

After the project, I could deeply sense that every solution is improved bit by bit with joint effort in different areas, like algorithm, image manipulation method, the sense of data and etc, and data-intensive computing is just one of the elements, specifically when the project scaling up. The project has deepened our understanding in big-data related project in a variety of fields. It is suggested to encourage the fellow students to attempt on more creative and interesting projects across different fields in the future.

9.2 Paroma Chatterjee

Contribution:

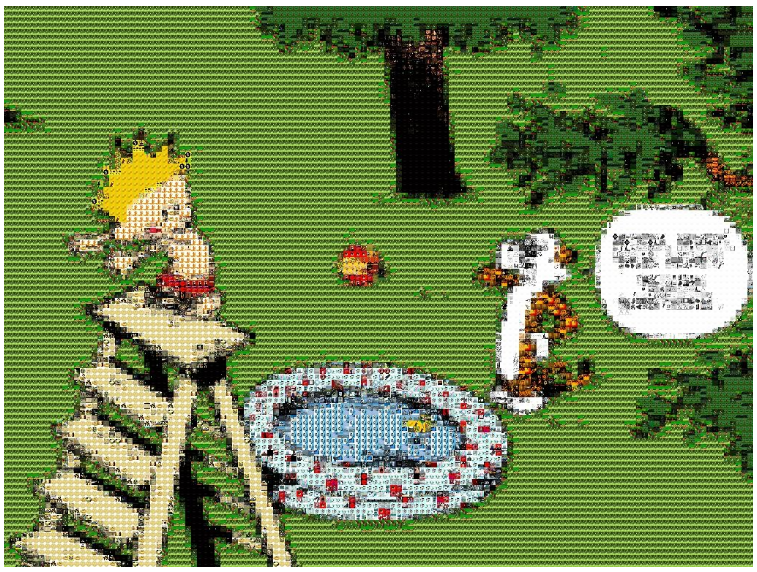
1. Suggested the topic
2. Brainstorming for the color bucket solution
3. Complete implementation of the color bucket solution
4. Helped with presentation slides and own part for report (map-reduce/ color bucket)
5. Collected test results for map-reduce part, collected results using Spark’s web UI
6. Managed repository on Git

Reflection:
For me, this project has been extremely challenging, especially to implement. I am glad that I could complete the working Color Bucket solution (mapreduce). Specifically, I learnt a lot about the different functions that Spark offers for its RDDs : the difference in use cases for flatMap and map, the different ways to collect an RDD into the driver's memory (collect and collectAsMap()) etc. I used a broadcast variable in my part and learnt what best practices must be followed while using such a variable.

In using Spark's input formats for all images, I learnt about the difficulties of converting from raw byte format to something I could work with (in this case it turned out to be numpy arrays because we were working in python). Unfortunately, I also realised a little too late how slow HDFS reads can be.

On the non-technical side, I was very lucky to have an amazing group of senior students who taught me how presentations and reports should be done properly. They were kind enough to be patient with me while I struggled with my code.

I already know that my solution is far from perfect, and have started thinking about obvious ways to tune the job. I will continue to apply the lessons from this project and course.



9.3 TSE Kwok Chiu Shayne

Contribution

1. Designed and Implemented the K-means clustering solution with Kennedy
2. Testing the K-means clustering solution and basic solution
3. Drawing all diagrams
4. Presentation slides
5. K-means clustering solution part in the Final Report

Reflection

This project is actually really challenging for us as especially it was my first time to process image data with programming. For a person that's used with C# for many years, Python is actually a really unfamiliar language for me. I always find some difficulties in implementing solution in Python during this project. Luckily, once again, I found that peer programming is really useful. The solution has been implemented successfully with limited amount of bug during testing stage as a benefit from peer programming with Kennedy.

After this topic, I found that understanding and playing with the image data is much more difficult and requires much more deeper knowledge than plain text data. For simply just manipulating the data pixel by pixel could not fully explore the potential of a image as each pixel and its' neighbourhood's pixel has some sense of relationship that missed in this project. From this project, I have learned and tried the potential and the beauty of data-intensive computing, and also deepen our understanding on this field via practical implementation. The experience and knowledge from this project would be really useful for us in our future career.

9.4 Ng Tsz Ho Dickson

Contribution

1. Dataset Collection
2. Presentation slides
3. Introduction and basic solution part in Final Report
4. Mirror coding for target image process part

Reflection

The project is most interesting and funny then other project for me. The topic of our project is mosaic image process. This topic is quite surprise for me because I consider to handle song or file dataset originally.

Before to start the topic, I have not any idea to handle the image process and larger dataset. This is the first time for me to do the image data-intensive approach project. Firstly, we are designing using the map-reduce solution to process larger image dataset and using python and spark language. Since my program skill is weakness than our group mate, I focus on pretension slide and data collection more than technique part.

However, when the group mate to discuss the different solution to our meeting, I learn more different efficiency solution. We find map-reduce is not enough because the time requirement still high, Kennedy and Shayne suggest using K-means for our second solution. They think K-means can faster than another to handle the larger image dataset.

After this topic, I learn more knowledge that the operation of k-means and mapreduce. Also, I understand the mosaic image process on python and spark language. Finally, I am thanks to my group mate suggest different solution and tech me many skill for presentation and compute the image process. I think the skill from the project is useful for my future.

10 Dataset and Code References

1. INRIA holidays dataset for image search.
   "Hamming Embedding and Weak geometry consistency for large scale image search" Proceedings of the 10th European conference on Computer vision, October, 2008
2. Basic for-loop solution : https://github.com/cinemast/OpenMosaic
3. Spark programming reference :
   a. https://spark.apache.org/docs/1.2.1/programming-guide.html
   b. https://spark.apache.org/docs/1.6.1/programming-guide.html
4. Numpy documentation : https://docs.scipy.org/doc/numpy-1.13.0/reference/

11 Repository

https://github.com/chatterburger97/map-reduce-mosaic