Cloud Anchor Extraction with Spark

CHENGLAI HUANG, University of California, Los Angeles, USA
JESSE HUANG, University of California, Los Angeles, USA
LIHANG LIU, University of California, Los Angeles, USA
RAGHAV DHALL, University of California, Los Angeles, USA

Augmented Reality is a popular technique nowadays. To enable interactions between multiple AR users from the same environment, a practical way is to extract the feature points from each users' view and spot the anchors to perform image stitching. This is a computation-intensive task, and to deal with this large computing power requirement, cloud computing can be applied. In this report, we simulate the AR interaction between multiple users by stitching together up to eight videos with different image stitching algorithms, and run the code on Google Cloud Platform to evaluate its performance.

ACM Reference Format:
Chenglai Huang, Jesse Huang, Lihang Liu, and Raghav Dhall. 2020. Cloud Anchor Extraction with Spark. In . ACM, New York, NY, USA, 7 pages.

1 INTRODUCTION AND RELATED WORK

Augmented Reality (AR) is an increasingly popular technique over the last few years [1]. While most AR applications are for a single user, we can combine multiple users' views and enable interaction between them if we want to enable AR collaboration. This involves image stitching techniques. For example, if we have two users in the same setting, we can get a frame from each user's view, and extract feature points from these two frames. If the users share a common object in their view, the object can be viewed as the anchors and used to establish the connection between the views.

This, however, is a very computation-expensive task. Feature extraction and matching is computation intensive by itself, not to mention that we are stitching every frame in the entire stream with ideally as little time needed as possible. Therefore, to achieve this goal, cloud computing techniques can be applied to improve throughput.

In this project, we simulated the AR interactions between multiple users by stitching together up to eight videos into a single one. More specifically, we first turned the videos into a set of frames, and stitch the groups of corresponding frames with image stitching algorithms. There are many existing related works we could utilize such as Sziliski's paper on introducing image stitching [6], and different feature detection techniques, such as SIFT [3], SURF [2], ORB [5], and those ones utilizing Machine Learning like YOLO [4]. We tried those different image stitching and feature extraction techniques and algorithms and chose the one with best performance. After we got the stitched frames, we turned them back into a final stitched video. We distributed the work to realize cloud computing with Google Cloud Platform with two different distribution algorithms, and compared their result with the single-server version.

In this report, section 2 discusses about the design of our image stitching model and methods used to run the model distributively on GCP in detail, section 3 shows a demo to our current model's result, section 4 evaluates our model's performance on GCP with cloud computing, and section 5 proposes some possible future work.

2 DESIGN

2.1 Stitching Algorithm

In our design of the stitching algorithm, we decided not to use OpenCV's stitcher class. This is because we realised that OpenCV's implementation has several restrictions on the input images and constantly failed to stitch images in undesired conditions (e.g. too little overlaps, too large angular changes). Additionally, implementing our own stitching algorithm gave us more control over the stitching process and allowed us to distribute each step of the stitching task in our Spark environment. Our main design objective is to support stitching of multiple frames (up to 8 frames) that is insensitive to the ordering of the frames. The pipeline includes three steps:

(1) Extracting scale invariant feature points from the frames
(2) Determine the ordering of frames
(3) Find the homographies between neighboring frames and stitch them

In step (1), after experimenting with different feature extraction methods, we decided that SIFT with number of keypoints limited to 500 provides the best performance and accuracy. In step (3), we used RANSAC for robust homography estimation. Both steps are implemented with OpenCV's libraries. Step (2) is introduced in 2.1.1.

2.2 Ordering of Frames

To find the ordering of frames, we used KNN matcher to match the keypoints in each pair of frames and store the the number of matching keypoints in a "match matrix". Figure 1 shows an example of a match matrix for 4 images.

	Image1	Image2	Image3	Image4
Image1(2)	0	250	**225**	100
Image2(1)	250	0	10	**25**
Image3(3)	225	10	0	**50**
Image4(4)	100	25	**50**	0

Fig. 1. Example Match Matrix

Our hypothesis was that a frame that should be in the centre of the stitched image will have two other frames that it matches well with. In contrast, an edge will have only one frame that it matches well with. Thus, we can examine the second highest number of matches of each image (shown in bold in Table 1), and consider the image with the lowest value to be an edge (image2). So, we appended this to the order and then found the image that it matches best with as its neighbor (image1) and appended the neighbor to the order. We continued this process till all the images were appended. The final ordering will be (image2, image1, image3, image4).

Next, we still need to check the direction of the ordering. To do this, we only need to check the relationship between the edge and its neighbor. First, we found the number of matches on the left side of the edge and the number of matches on the right side of the neighbor. Then we calculated the ratio of these points to the total number of matches. If this ratio is greater than 50%, it means the neighbor is to the left of the edge, so the entire ordering should be reversed. Otherwise, the ordering should remain the same.

2.3 Distributive Algorithm

Given N videos each having T frames, we need to process $N \times T$ images. Naturally, there are two design choices for distribution: (1) distribute over the number of videos (N); (2) distribute over the number of frames (T). In the following section, we will discuss the implementation and analysis of both approaches.

2.3.1 Distribute-over-N. Distribute-over-N means that in each Spark job, we stitch N frames distributively by extracting one frame from each of the N videos. Then, we run T of those Spark jobs sequentially to process the entire videos. Our initial motivation to use distribute-over-N was because the stitching of N frames takes a long time and we believe distribution can significantly reduce the time.

Feature Extraction. In this step, we read an RDD of images as binary files, and map the binary-file-to-image conversion function and the keypoint extraction function to the RDD.

Match Matrix Calculation. This step is more complicated because we need to covnert the 1D list of keypoints into a 2D matrix of matches. Initially, we called the cartesian() method to the RDD itself to create a matrix for calculation. We also used zipWithIndex() method to assign an index to each image to prevent repeated calculation of $match(i, j)$ and $match(j, i)$.

However, this method turns out to be inefficient, likely due to the huge cost of zipWithIndex() and cartesian() method. Hence, we developed the more efficient second method, where we store all unique (i, j) pairs to calculate into an RDD, then map the matching function to the RDD, and finally convert the result to a 2D matrix.

Ordering and Stitching. Both ordering and stitching involve sequential processing and need to check individial entries in the match matrix. Because Spark only supports coarse-grained processing, we have to collect() the images and matchings from the RDDs to the master node, which is costly. Therefore, as shown the in the evaluation in 4.2, distribute-over-N is inefficient overall.

2.3.2 Distribute-over-T. The inefficiencies of distribute-over-N motivated us to move to distribtue-over-T. Intuitively, it also makes more sense to distribute over T because the size of T is much larger than the size of N. Moreover, while exploring distribute-over-N, we managed to reduce the time to stitch the N frames by optimizing code and adopting better feature extraction methods, so our original concern no longer existed.

Distribute-over-T means to process T stitching jobs distributively, and each stitching job of N frames is done in a single node approach. Since each stitching job becomes a unit of computation, we avoided calling collect() in the middle as in distribute-over-N. As illustrated in 4.3, this approach is significantly more efficient than single-node approach and distribute-over-N.

The implementation of distribute-over-T is also simpler than distribute-over-N. We read all frames of all videos into an RDD of a $T \times N$ matrix, where each row in the matrix is the N frames to stitch. Then, we apply the stitching function to each row in the RDD, and the results collected would be the stitched images.

3 RESULT

4 EVALUATION

For the evaluation of algorithms, we set up Spark clusters on Google Cloud Platform using Dataproc service. Section 4.1 describes the experiment configuration and out datasets, 4.2 and 4.3 discuss the performance comparison of distribute-over-N and distribute-over-T, respectively.

4.1 Expriment Environment and Datasets

Dataproc is a service of Google Cloud Platform that provides an easy-to-use interface for setting up Hadoop/Spark clusters in a few clicks. As Dataproc has a limit of 8 vCPUs for a free trial account, we set up clusters with two configurations, as figure 2 , under this restriction. 2-worker has 2 worker nodes, each with 2 vCPUs on Intel Gen 6th Skylake CPU, 7.5 GB memory and a 500GB hard disk storage, while 3-worker has 3 worker nodes with the same configuration. The only difference would be the master node of the 2-worker cluster has 4 vCPU and 15GB memory. Master and worker nodes are allocated within a datacenter connected by local area ethernet network, the maximum network bandwidth is 10Gbps.

	2-worker	3-worker
master	n1-standard4 (4 vCPUs, 7.5GB)	n1-standard2 (2 vCPUs, 7.5GB)
worker	n1-standard2 (2 vCPUs, 7.5GB)	n1-standard2 (2 vCPUs, 7.5GB)

Fig. 2. Cluster Configuration

For distribute-over-N, We use the datasets in figure 3 for the following experiments, those frames are extracted from random videos taken from the internet. Sample 1 contains 6 different frames

Sample No.	format	frames	resolution	size
1	PNG	6	4032x3024	1̃0MB per
2	JPEG	6	667x768	9̃50KB per
3	PNG	8	318x772	3̃00KB per

Fig. 3. Datasets

4.2 Distribute-over-N

We first evaluate the performance of our first approach, which is the Distribute-over-N, to our baseline single-server version. Because the workflow of two versions are different, the whole process is separated into frame extraction, matching and stitching. We run the single-server version on the master node of the 2-worker cluster.

The chart in figure 4 shows that the performance of Spark version is even worse than the single-server version, this is due to our unoptimized code and inefficient job splitting. For the matching part, the Spark version has some improvement, and it is 2 times faster in some cases. And two versions take nearly the same amount of time on performing the stitching, thus there is no improvement at stitching. Although the Spark version indeed runs faster on the matching part, the improvement is not significant enough to remedy the performance lost on frame extraction.

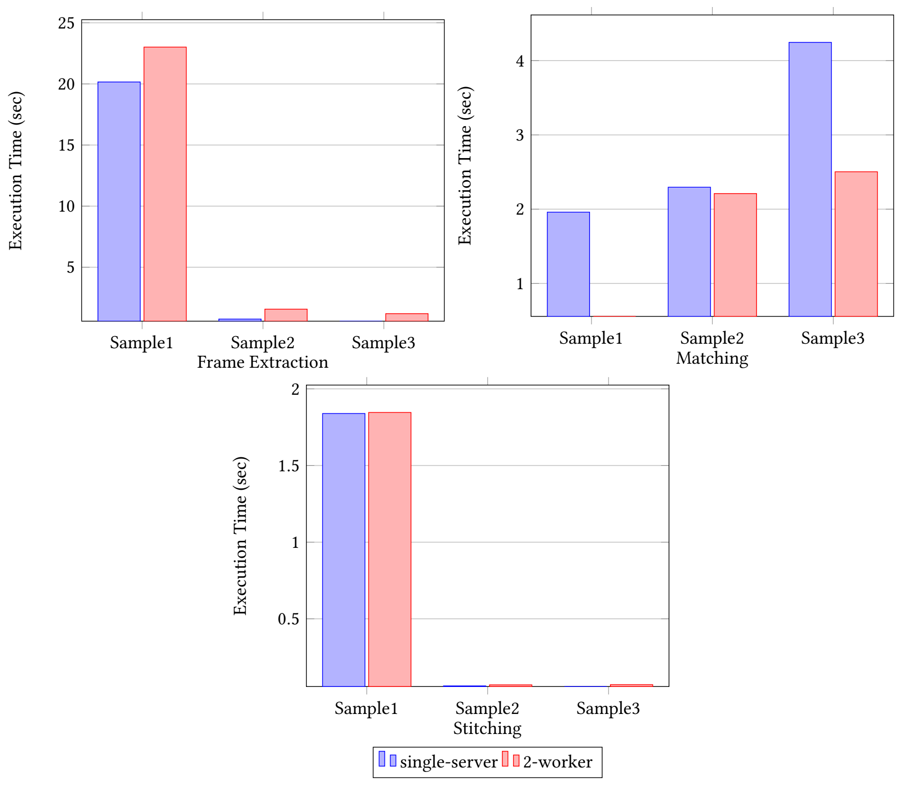

Fig. 4. Performance of distribute-over-N

4.3 Distribute-over-T

Apparently, our first approach is not beneficial, hence we finally came up with the second approach, Distribute-over-T.

The Spark code using the Distribute-over-T approach is actually replicating the code of the single-server version to run on several nodes, thus we do not split the process for comparison here. For this part, the input data are our sample videos of the shoe organizer, which have 2 videos each with 180 frames.

First, we compare the performance of different number of worker nodes. As shown in figure 5 ,this time, the Spark version has notably reduce the run time, up to around 70% with 3 worker nodes, compare to the single-server version. Next, we compare the performance of single-server and 3-worker regarding different number of frames and the results are also shown in figure. Clearly, the run time of both versions grow linearly, but 3-worker has a smaller factor and grows slower, as the number of frames increases.

5 FUTURE WORK

In this section, we briefly talk about some of the possible improvements that could be applied to the current model.

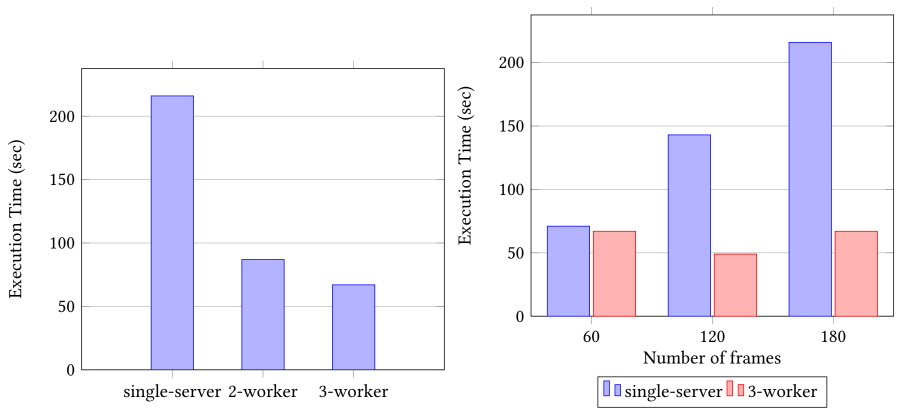

Fig. 5. Performance of distribute-over-T

5.1 Enable Real-time Video Stitching

In practice, it doesn't make much sense for the user to upload their videos and wait for it to be processed in AR applications. Therefore, we could set up a streaming protocol to realize in-time video stitching. To achieve this goal, several methods can be applied. For example, we can increase the number of worker nodes in the cloud computing system. Real-time video stitching would largely increase computing intensity, and it's necessary to set up more worker nodes for the task. However, the quota and disk restrictions on GCP doesn't allow us to do so for the current model. In addition, we could also process the frames by smaller batches to increase throughput.

5.2 Improve Image Stitching Model

In previous sections, we showed and explained why our current image stitching model is still not optimal and resulting in glitches in the output. To solve this problem, we should investigate our current image stitching model to detect any problems. Though the algorithm we used should be sufficient for our task, we could further improve it with different optimization techniques, such as moving DLT [7] or minimizing false edges [8].

6 CONCLUSION

In this project we simulated interaction between AR users by stitching multiple videos using different feature detection and image stitching algorithms. We decided to implement our own model rather than directly using the OpenCV stitcher library to get rid of the restrictions and gain more control. For the model, we chose SIFT with a keypoint threshold of 500 after comparing different algorithms. Though the demo result has some glitches, it shouldn't be a large issue. A possible explanation is that there's not enough significant feature points in our test video, and some optimization to the model should solve the issue. Then, we ran the task on Spark by deploying to Google Cloud Platform. We chose to distribute over T rather than over N, and achieved promising results compared to the single node version as per the evaluation. Lastly, we proposed some possible improvements, which includes setting up a streaming protocol and optimizing our current image stitching model.

ACKNOWLEDGMENTS

Acknowlegement

REFERENCES

[1] Mark Billinghurst, Adrian Clark, and Gun Lee. 2015. A Survey of Augmented Reality. *Found. Trends Hum.-Comput. Interact.* 8, 2–3 (March 2015), 73–272. https://doi.org/10.1561/1100000049

[2] L. Juan and G. Oubong. 2010. SURF applied in panorama image stitching. In *2010 2nd International Conference on Image Processing Theory, Tools and Applications*. 495–499. https://doi.org/10.1109/IPTA.2010.5586723

[3] David G. Lowe. 2004. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision* 60, 2 (Nov. 2004), 91–110. https://doi.org/10.1023/B:VISI.0000029664.99615.94

[4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. 2016. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 779–788. https://doi.org/10.1109/CVPR.2016.91

[5] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. 2011. ORB: An efficient alternative to SIFT or SURF. In *2011 International Conference on Computer Vision*. 2564–2571. https://doi.org/10.1109/ICCV.2011.6126544

[6] Richard Szeliski. 2006. Image Alignment and Stitching: A Tutorial. *Found. Trends. Comput. Graph. Vis.* 2, 1 (Jan. 2006), 1–104. https://doi.org/10.1561/0600000009

[7] J. Zaragoza, T. Chin, M. S. Brown, and D. Suter. 2013. As-Projective-As-Possible Image Stitching with Moving DLT. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 2339–2346. https://doi.org/10.1109/CVPR.2013.303

[8] A. Zomet, A. Levin, S. Peleg, and Y. Weiss. 2006. Seamless image stitching by minimizing false edges. *IEEE Transactions on Image Processing* 15, 4 (2006), 969–977. https://doi.org/10.1109/TIP.2005.863958