LiveTraj: Real-Time Trajectory Tracking over Live Video Streams

Tom Z. J. Fu[1] Jianbing Ding[1, 2] Richard T. B. Ma[3, 1] Marianne Winslett[1, 4]
Yin Yang[5, 1] Zhenjie Zhang[1] Yong Pei[1] Bingbing Ni[1]
[1]Advanced Digital Sciences Center, Illinois at Singapore Pte. Ltd.
[2]School of Information Science and Technology, Sun Yat-sen University
[3]School of Computing, National University of Singapore
[4]Department of Computer Science, University of Illinois at Urbana-Champaign
[5]College of Science and Engineering, Hamad Bin Khalifa University
{tom.fu, jianbing.d, tbma, winslett, yin.yang, zhenjie, pei.yong, bingbing.ni}@adsc.com.sg

ABSTRACT
We present LiveTraj, a novel system for tracking trajectories in a live video stream in real time, backed by a cloud platform. Although trajectory tracking is a well-studied topic in computer vision, so far most attention has been devoted to improving the *accuracy* of trajectory tracking, rather than the *efficiency*. To our knowledge, LiveTraj is the first that achieves real-time efficiency in trajectory tracking, which can be a key enabler in many important applications such as video surveillance, action recognition and robotics. LiveTraj is based on a state-of-the-art approach to (offline) trajectory tracking; its main innovation is to adapt this base solution to run on an elastic cloud platform to achieve real-time tracking speed at an affordable cost.

The video demo shows the offline base solution and LiveTraj side by side, both running on a video stream containing human actions. Besides demonstrating the real-time efficiency of LiveTraj, our video demo also exhibits important system parameters to the audience such as latency and cloud resource usage for different components of the system. Further, if the conference venue provides sufficiently fast Internet connection to our cloud platform, *we also plan to demonstrate LiveTraj on-site*, during which we will show LiveTraj identifying and tracking trajectories from a live video stream captured by a camera.

Categories and Subject Descriptors
I.2.10 [**Artificial Intelligence**]: Vision and Scene Understanding – *video analysis*; I.3 [**Computer Graphics**]: Hardware Architecture – *parallel processing.*

General Terms
Performance, Design

Keywords
Real-time video analysis, trajectory tracking, elastic cloud.

MM'15, October 26–30, 2015, Brisbane, Queensland, Australia.

DOI: http://dx.doi.org/10.1145/2733373.2807401

1. INTRODUCTION
We present LiveTraj, a real-time trajectory tracking system over live videos, running on an elastic cloud platform. Unlike the majority of existing trajectory tracking systems, LiveTraj features real-time processing, which means that its throughput must keep up with the frame rate of the video stream at all times. This capability is critical in many applications, including video surveillance, robotics, autonomous vehicles, etc. For instance, LiveTraj can be deployed to extract human body movements from a surveillance video feed; such movements can be further analyzed to recognize public security threats as well as socially undesirable actions such as fighting. Additionally, LiveTraj can be used to monitor people flows in a crowded area such as a city square during a major festival, which helps prevent accidents such as stampedes, such as the one happened in Shanghai on the New Year's Eve in 2014[1].

One main technical challenge in building LiveTraj is that the workload for video trajectory tracking can be highly dynamic, depending on the content of the video. Figure 1 shows two snapshots from our demo video. In Figure 1a, a man is clapping his hands; clearly, there are only a small number of the extracted trajectories (shown in green lines), which concentrate on the part of the video frame centering on his hands. In contrast, in Figure 1b, where two people are walking, LiveTraj identifies and tracks a much larger number of trajectories. As we explain in Section 3, the workload of LiveTraj depends on the number of trajectories being tracked; hence, the frame shown in Figure 1b incurs significantly higher workload compared to that in Figure 1a. Since LiveTraj runs in real-time, the former frame (Figure 1b) also requires more computational power than the latter (Figure 1a) in order to keep up with the video frame rate. In general, the amount of work can vary from frame to frame in an unpredictable manner, depending on the number of detected trajectories. Provisioning computational resources based on the peak workload would lead to considerable idle time and, thus, waste of resources.

Our solution in LiveTraj is to employ an elastic cloud system to support such fluctuating workload. In particular, the backend of LiveTraj is based on Resa [12], a cloud-based real-time stream processing system built by the authors. Resa includes DRS [5], a dynamic resource scheduler that optimally allocates cloud resources such as CPU cores to different system components, such that overall resource usage is minimized, while ensuring real-time response. Since most cloud providers have a pay-as-you-go

[1] http://en.wikipedia.org/wiki/2014_Shanghai_stampede

pricing scheme, such optimized resource allocation effectively reduces the financial cost for running a real-time trajectory monitoring system. Additionally, Resa also contains a component for efficiently migrating workload between different nodes in a cluster, which facilitates dynamic node additions / removals, and ensures load balancing [3].

(a) Low workload frame (b) High workload frame

Figure 1. Examples of real-time trajectory tracking

LiveTraj is based on an existing state-of-the-art offline trajectory tracking algorithm [15], which achieves high accuracy. In fact, two of our team members have won the ICPR-HARL 2012 competition[2] using an improved version of this algorithm [9]. The main idea is to start new trajectories at dense points [14], and tracks trajectories using optic flows [11]. Note that adapting this algorithm to the cloud platform is non-trivial, as it involves complex data flows containing a loop, as elaborated in Section 3. Further, LiveTraj reduces communication costs between different nodes in the cloud through novel partitioning schemes, avoids redundant computations, and facilitates balanced workloads in different nodes. These designs help LiveTraj achieve real-time response using only a small amount of cloud resources. Our video demo shows that LiveTraj processes a video stream with 640×480 resolution in real time, using merely 7 commodity nodes.

We have prepared both a video demo and an on-site demo. The video demo compares LiveTraj side-by-side with its base algorithm [15], which is so slow that its output video barely moves. Additionally, our video demo also monitors key system performance measurements, e.g., the response time and resource consumption of different system components. The on-site demo will also show the effectiveness and efficiency of LiveTraj at analyzing a live video stream captured by a camera. A pre-requisite of the on-site demo is that the conference venue provides sufficiently fast Internet connection to our cloud platform, so that the live video can be streamed in real time to the cloud servers that run LiveTraj, and at the same time the results of LiveTraj can be streamed back for on-site presentation.In the following, Section 2 overviews related systems. Sections 3 presents LiveTraj. Section 4 describes our video demo as well as the proposed on-site demo. Section 5 concludes with directions for future work.

2. BACKGROUND

Offline video trajectory tracking. Tracking trajectories in an input video is a well-studied problem in computer vision. In the following we overview the offline tracking algorithm [15] used in LiveTraj, which is a state-of-the-art method for video trajectory tracking. Figure 2 illustrates the major components of this method. Specifically, each input video frame is analyzed by two independent components: dense point extraction and optic flow generation. A dense point [14] marks the beginning of a new trajectory, where as an optic flow [11] tracks the movements of pixels in the video frame. After that, the main trajectory tracking module generates trajectories by combining the dense points, the current set of tracked trajectories, and the optic flows.

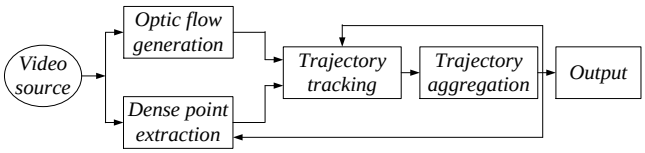

Figure 2. Offline video trajectory tracking algorithm [15]

In particular, for each dense point, the algorithm starts a new trajectory, and later on uses the optic flow to compute its next point. Similarly, for each existing trajectory, the tracking module uses the optic flow to check whether the trajectory has ended, and, if not, the position of the next point on the trajectory. Finally, the generated trajectories are aggregated (e.g., to remove duplicates) and reported to the user. Meanwhile, the results are fed back to the dense point extraction module (e.g., to eliminate dense points on existing trajectories) as well as the trajectory tracking module.

Cloud-based stream processing. Recently, several real-time, cloud-based stream processing systems have been built for applications involving "fast data", which include Apache Storm [13] and its successor Twitter Heron [7], Apache S4 [10], Apache Samza [1], StreamCloud [6] and Apache Spark Streaming [16]. Our Resa system [12] builds on top of Apache Storm, and adds various elasticity features. In particular, a key component of Resa is DRS [5], which, along with a dynamic workload migration module [3] and a resource negotiator called Abacus [17], provides support for *elastic resource allocation*, meaning that resources can be dynamically added to or removed from an application to match its current workload. Further, the resource allocation in DRS is optimized to minimize total resource usage, while satisfying the real-time response constraint. LiveTraj is based on Resa and uses DRS to manage resource allocations.

3. LIVETRAJ

LiveTraj adapts its base trajectory tracking algorithm described in Section 2 to run on an elastic cloud platform. Our main design goals are to (i) minimize redundant computations, (ii) reduce communication costs between nodes in the cloud, (iii) allow a high degree of parallelism and (iv) enable dynamic node additions and removals to match the current workload. Specifically, LiveTraj is based on Resa (described in Section 2), an enhanced version of Apache Storm. Each video manipulation operation in the algorithm, shown in Figure 2, is implemented as a *bolt* (i.e., logical operator) executed by multiple nodes, while the input frame arrive at the system via a *spout* (i.e., streaming source). The output of the system is presented on screen using FFmpeg[3].

We first focus on the dense point extraction bolt, which has two input streams: the input video frame and the current trajectories (from the trajectory aggregation operator shown in Figure 2). The output of this operator consists of dense points sampled in the video frame that are not already on any of the current trajectories. According to Ref. [15], to identify dense points, the algorithm uniformly samples coordinates in the input video frame, and then tests each of these coordinates based on the brightness of the corresponding pixel in the input frame. Figure 3 illustrates the parallel execution of the dense point extractor. In particular, LiveTraj partitions the frame into different regions, and assigns one region to a *dense point evaluator*, each running in a separate thread. Then, the sampled coordinates are grouped according to the partitioning, and routed to the corresponding dense point evaluator. Meanwhile, coordinates on current

[2] http://liris.cnrs.fr/harl2012

[3] https://www.ffmpeg.org

trajectories are similarly grouped by a *point dispatcher*, and routed accordingly. Such partitioning and routing minimizes network transmissions as each node is only fed the pixels and trajectory points it needs. To balance the workload of different nodes, we set the number of regions (a tunable parameter) to be larger than the number of nodes for the operator, and assign each node multiple regions. This design avoids stragglers (i.e., slow nodes forming the bottleneck of the entire system) caused by very dense regions, as the workload is partitioned in finer units. Finally, as the dense point evaluators are stateless, nodes can be added to or removed from the operator at run time, enabling dynamic elasticity provided by the underlying Resa platform.

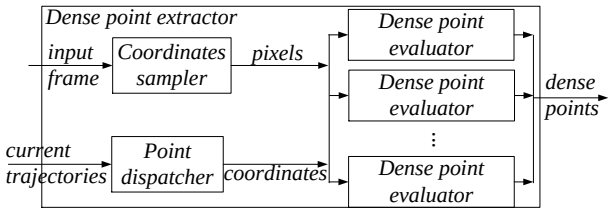

Figure 3. Parallel execution of dense point extraction

The optic flow generation operator is executed by multiple nodes in parallel similarly to the dense point extractor. An additional challenge here is that the generation of optic flows involves (i) comparing two frames at consecutive time instances and (ii) multiple pixels in determining the value of the flow in each coordinate. (i) means that the operator is *stateful*, i.e., each node must store the previous frame and compare with the current one. Hence, node additions and removals (necessary for elasticity) become non-trivial as a new node does not immediately possess the necessary states to work (i.e., pixels on the previous frame) on its inputs. Fortunately, these are handled by the DRS module of the underlying Resa platform, which performs operator state migrations transparently [3][5]. Regarding (ii), each node cannot simply handle a region in the frame, as is the case in the dense point extractor, as the computation at one coordinates relies on the surrounding pixels. A naive solution is to broadcast the entire input frame to each node, which, however, incurs a high communication overhead. Our solution in LiveTraj is to split the frame into overlapping *patches* [8]; each patch contains a partition of the frame, as well as the pixels surrounding the partition, as illustrated in Figure 4. This design effectively reduces the amount of network transmissions, thus improving system scalability.

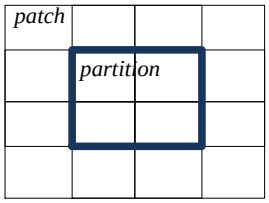

Figure 4. Example of frame partition and patch

Finally, we examine the trajectory tracking operator, which involves three inputs: the current trajectories, the dense points detected from the input frame, and the optic flows of the input frame. The main idea of this operator is to "grow" a trajectory, either an existing one or a new one starting at a dense point, by adding one more coordinate computed from the optic flow. Note that it is possible that the optic flow indicates that there is no more coordinate on this trajectory in the input frame, ending the trajectory. The parallelization of this operator is similar to that of the dense point extractor, except that each node is assigned trajectories rather than pixels and coordinates. Grouping of the trajectories is performed according to their last coordinates (or the newly identified dense points for new trajectories). We omit further details for brevity. Observe that the workload of this operator depends on the number of extracted dense points and current trajectories: when there are numerous trajectories being tracked (e.g., in Figure 1b), the system has heavier workload, which may necessitate node additions. Conversely, when there are few trajectories (Figure 1a), the amount of workload drops and the system can release some nodes. Our elastic Resa platform handles such resource changes efficiently and effectively, which reduces resource costs while ensuring real-time response.

4. DEMONSTRATION SCENARIO

Video demo. Figure 5 shows a screenshot of our video demo. The panel on the left monitors important performance measurements of different components of the system, e.g., dense point extraction and optic flow computation. The performance indicators include response time, resource consumption, etc. In the center we present the output video streams of both the offline algorithm and LiveTraj side by side. The output video of LiveTraj plays smoothly, whereas the output for the offline algorithm barely moves at all. Finally, the bottom of the demo video displays the command lines and textual outputs of the two systems, which inform the audience of their status updates.

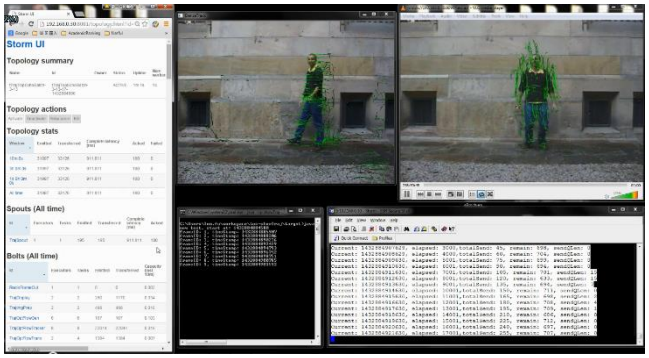

Figure 5. Screen shot of the video demo

The video demo was recorded while LiveTraj was running on 8 nodes, each equipped with an Intel quad-core 3.4GHz CPU and 24GB RAM. Following common practice, we dedicate one node as the master node, running Nimbus (the master node of Apache Storm as well as Resa) and Apache Zookeeper server [2], used in Storm and Resa. Hence, there are 7 worker nodes in total. The input video is from the ChaLearn Looking at People 2014 Challenge [4], which has a resolution of 640×480, at 30 frames per second. The input video is specifically designed for human pose, gesture and action recognition [4]. Note that given sufficient resources, LiveTraj could handle video streams with much higher resolutions. We chose this one due to limitations of the file size in the submission system.

On-site demo. The planned on-site live demo uses LiveTraj to track trajectories of an arbitrary input stream captured by a camera. For instance, the audience can stand in front of the camera and perform arbitrary body movements, and LiveTraj will mark these movements in real time. The audience can then observe the effectiveness (i.e., accuracy) and efficiency (i.e., in terms of output smoothness and result delay) of LiveTraj. Note that the onsite demo is only possible when the conference venue provides sufficiently fast Internet connection, since all computations will be performed remotely on our cloud servers. For the same reason, the resolution of the input video will be tuned according to the Internet speed at the conference site.

5. CONCLUSIONS AND FUTURE WORK

We propose to demonstrate LiveTraj, a real-time video trajectory tracking system running on an elastic cloud platform. LiveTraj is based on a state-of-the-art offline trajectory tracking algorithm using dense points and optic flows, and it adapts this algorithm to the elastic cloud environment through a number of innovative designs as well as the underlying Resa platform. Our video demo compares the efficiency of LiveTraj with the offline base solution, and presents to the audience the smooth and low-lag output stream of the former. Given sufficiently fast Internet connection at the conference venue, we will also show an onsite demo that further enriches the audience's experience by letting them try LiveTraj using a live video stream captured on site.

6. ACKNOWLEDGEMENTS

This study is supported by the research grant for the Human-Centered Cyber-physical Systems Programme at the Advanced Digital Sciences Center from Singapore's Agency for Science, Technology and Research (A*STAR). Ding is partially supported by NSF of China under Grant 61173081, and also supported by GZSI's grant 2013Y2-00046.

7. REFERENCES

[1] Apache Samsa. *http://samza.apache.org* .

[2] Apache Zookeeper. *https://zookeeper.apache.org/* .

[3] Ding, J., Fu, T., Ma, R., Winslett, M., Yang, Y., Zhang, Z., Chao, H. Optimal Operator State Migration for Elastic Data Stream Processing. *CoRR*, 1501.03619, 2015.

[4] Escalera, S., Baró, X., Gonzàlez, J., Bautista, M., Madadi, M., Reyes, M., Ponce-López, V., Escalante, H., Shotton, J., Guyon, I. ChaLearn Looking at People Challenge 2014: Dataset and Results. In *Proc. ECCV Workshops*, 2014.

[5] Fu, T., Ding, J., Ma, R., Winslett, M., Yang, Y., Zhang, Z. DRS: Dynamic Resource Scheduling for Real-Time Analytics over Fast Streams. In *Proc. IEEE ICDCS*, 2015.

[6] Gulisano, V., Jiménez-Peris, R., Patiño-Martínez, M., Soriente, C., Valduriez, P. StreamCloud: an Elastic and Scalable Data Stream System. *IEEE TPDS*, 23(12): 2351-2365, 2012.

[7] Kulkarni, S., Bhagat, N., Fu, M., Kedigehalli, V., Kellogg, C., Mittal, S., Patel, J., Ramasamy, K., Taneja, S. Twitter Heron: Stream Processing at Scale. In *Proc. ACM SIGMOD*, 2015.

[8] Lindeberg, T. Scale Invariant Feature Transform. *Scholarpedia*, 7(5): 10491, 2012.

[9] Ni, B., Pei, Y., Moulin, P., Yan, S. Multi-Level Depth and Image Fusion for Human Activity Detection. *IEEE TSMC (B)*, 43(5): 1383-1394, 2013.

[10] Neumeyer, L., Robbins, B., Nair, A., Kesari, A. S4: Distributed Stream Computing Platform. In *Proc. IEEE ICDM KDCloud Workshop*, 2010.

[11] O'Donovan, P. Optical Flow: Techniques and Applications. Technical Report 502425, University of Saskatchewan, 2005.

[12] Tan, T, Ma, R., Winslett, M., Yang, Y., Yong, Y., Zhang, Z. Resa: Realtime Elastic Streaming Analytics in the Cloud. In *Proc. ACM SIGMOD*, 2013, poster.

[13] Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J. M., Kulkarni, S., Jackson, J., Gade, K., Fu, M., Donham, et al. Storm@Twitter. In *Proc. ACM SIGMOD*, 2014.

[14] Wang, H., Kläser, A., Schmid, C., Liu, C.-L. Action Recognition by Dense Trajectories. In *Proc. CVPR*, 2011.

[15] Wang, H., Kläser, A., Schmid, C., Liu, C.-L. Dense Trajectories and Motion Boundary Descriptors for Action Recognition. *IJCV*, 103(1): 60-79, 2013.

[16] Zaharia, M., Das, T., Li, H., Shenker, S., Stoica, I. Discretize Streams: an Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters. In *Proc. ACM SOSP*, 2013.

[17] Zhang, Z., Ma, R., Ding, J., Yang, Y. ABACUS: An Auction-Based Approach to Cloud Service Differentiation. In *Proc. IEEE IC2E*, 2013.

[18] Zhang, Z., Shu, H., Chong, Z., Lu, H., Yang, Y. C-Cube: Elastic Continuous Clustering in the Cloud. In *Proc. IEEE ICDE*, 2013.