An Edge Datastore Architecture For Latency-Critical Distributed Machine Vision Applications

Arun Ravindran and Anjus George
Department of Electrical and Computer Engineering
University of North Carolina at Charlotte

Abstract

Multi-camera real-time vision at the Edge is facilitated by low-latency distributed data stores. In this paper, we take the position that latency criticality in the challenging operating conditions at the Edge can only be attained through application specific designs incorporating autonomous computing techniques. In our initial prototype, we implement a key-value Edge data store that autonomously monitors run-time conditions to maintain latency-criticality of one class of data (feature vectors), while sacrificing the latency and accuracy of another class of data (keyframes). Early results show a median latency improvement of 84.8% over non-autonomous operation, for videos with large scene dynamics, and operational conditions of intermittent wireless channel interference.

1 Introduction

The recent emergence of powerful Deep Learning algorithms, along with the ability to store and process massive amounts of data, has given us the ability to potentially recognize objects in near real-time [13]. Such real-time machine vision is a foundational technology in a number of applications such as automatic video surveillance, augmented and virtual reality, and vision assisted robots. In many of these applications, timely recognition of objects and their activity is important since events need to be responded to within tight deadline constraints. The latency critical nature of machine vision applications motivates the use of the Edge computing paradigm [22][14][21][20][25][23][7][8]. While single cameras have been used for machine vision applications, increasingly researchers are investigating the use of multiple cameras for distributed video analytics. In scenarios such as surveillance in dense urban environments, where occlusions are common, multiple camera views increases tracking robustness [26]. Vision analytics applications at the Edge aggregates data from multiple camera nodes to perform trajectory and behavioral analysis. A data storage abstraction at the Edge is a key system component that facilitates development of such analytics applications.

A number of Cloud storage systems have been proposed over the last decade [9][12][6][4][18]. While Edge and Cloud systems share similar features such as the need for scalability, and fault tolerance, significant differences exist.

1. Cloud systems are housed in pristine datacenters. On the other hand, Edge systems are often deployed in the "field" where highly dynamic operating conditions exist. For example, the nodes most likely use a wireless communication link operating in unlicensed bands, where significant intermittent interference exists from other users.

2. The data is inherently distributed at the Edge nodes due to the distributed nature of the data sources (for example, cameras). This is different from the Cloud where the data is distributed by design across multiple nodes to accommodate large data sizes.

3. The heterogeneity of the storage nodes at the Edge is far more diverse than the Cloud. From a storage perspective, the embedded boards (at

the camera) has GB of storage, the Edge nodes (at access points/base stations) offer TB of storage, and the backend Cloud offers PB of storage.

4. The physical insecurity of the nodes in the field and the use of the Edge in critical cyberphysical systems brings additional security challenges to the Edge.

In this paper, we present early results from our investigation of a storage architecture that can potentially meet the needs of distributed vision analytics at the Edge. Our position is that the challenging performance requirements at the Edge is best addressed by (a) designing the storage architecture specifically for the application - in our case, multi-camera machine vision, and (b) employing system techniques from autonomic computing to meet performance metrics under dynamic operating conditions. In our initial prototype, we implement a key-value Edge data store that autonomously monitors run-time conditions to maintain latency-criticality of one class of data (image feature vectors), while sacrificing the latency and accuracy of another class of data (keyframes). Early results show a median latency improvement of 84.8% over non-autonomous operation, for videos with large scene dynamics, and WiFi operational conditions of intermittent channel interference.

The paper is organized as follows - Section 2 briefly introduces recent work on designing data stores for the Edge. Section 3 provides a description of the system architecture for distributed machine vision at the Edge. Section 4 identifies latency control knobs, while Section 5 presents the design of the data store. Section 6 reports the results. Section 7 discusses future research directions, and concludes the paper.

2 Related Work

In the VisFlow project, Lu et al. [16] describe a system that can analyze feeds from multiple cameras. In particular, they describe a dataflow platform for vision queries that is built on top of the SCOPE dataflow engine, that offers general SQL syntax, and supports added user-defined operators such as extractors, processors, reducers and combiners. In the Cachier project, Drolia et. al. [10] propose a caching system for the Edge that caches Deep learning models available on the Cloud on to the Edge nodes to take advantage of spatio-temporal locality of user access, thus minimizing the number of requests that go to the Cloud. In the CloudPath project, Mortazavi et. al [17] propose Path Computing as a new paradigm that generalizes the Edge computing vision into a multi-tier Cloud architecture deployed over the geographic span of the network. CloudPath has a distributed eventual consistent storage system (PathStore) that replicates application data on-demand. The above projects can be considered complementary to our work; our focus is on enabling latency critical operation of Edge data stores through autonomous computing.

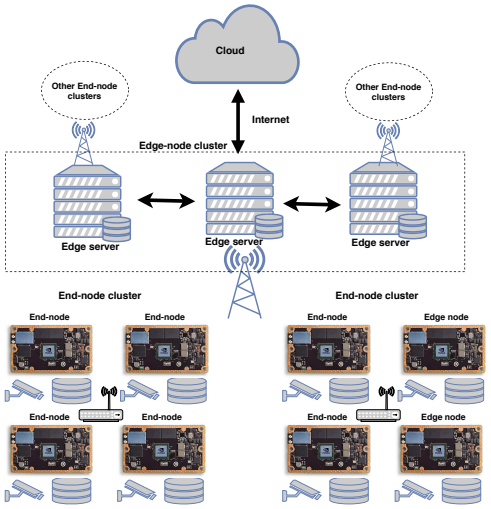

Figure 1: Distributed machine vision at the Edge

3 System Architecture

Figure 1 illustrates the Edge computing architecture for distributed machine vision. At the lowest level, a number of End-nodes equipped with cameras collectively monitors a geographic region (for example, a neighborhood) to determine events of interest (for example, drunk driving). A cluster of End-nodes (of-

ten based on proximity) is served by an Edge-node with communication over a wireless link (cellular or WiFi). Depending on the scale of the system, a cluster of such Edge-nodes may be served by a Core-node (not shown), and so on, with the Cloud at the top-most level. The End-nodes are assumed capable of one-hop communication to the Edge-node, with the Edge-node connected to the Cloud over the Internet. Physically, the End-nodes are located in the field (for example, traffic signal pole), the Edge-node, in a more secure location adjacent to the End-node (for example, traffic signal box), and the Cloud is hosted in a data center. We assume that the nodes at the higher levels of the hierarchy have more resources (compute, storage, network bandwidth, energy) available to them. Also, we assume that latency from the cameras increases as we go up the hierarchy from the End-nodes to the Cloud.

Each End-node is equipped with a powerful embedded computing board that has multiple processing engines including CPUs, GPUs, and custom FPGA based accelerators capable of processing video frames at real-time using compute-intensive vision algorithms (often involving Deep Learning) to extract image feature vectors (N-dimensional vector of numerical features that represent objects in an image), and key-frames (images with maximal number of feature vectors) [24]. The feature vectors and key-frames are transfered from the End-node to the Edge-node. Note that the feature vectors are typically 100x smaller in size than the key-frames. However, the feature vectors are generated at a higher rate (typically 30 fps) compared to key-frames (typically 1 fps). The Edge-node aggregates the feature vectors, from multiple End-nodes, to perform analytics (for example, behavioral analysis) across space and time depending on the event of interest. The Edge-node also aggregates the key-frames from the End-nodes both for archival purposes (for example, legal evidence), and to obtain any information not contained in the feature vectors extracted by the End-node. The data store operations involving the feature vectors is considered latency critical, while those involving key-frames is considered latency sensitive. Additionally, since the key-frames primarily serve archival purposes, we assume some loss of accuracy can be tolerated.

4 Latency Control Knobs

The primary component of the Edge data store latency is the data transfer operation from the End-node to the Edge-node over the wireless link. For WiFi, this latency could be on the order of tens of milliseconds [11]. Additionally, due to crowding in the unlicensed WiFi spectrum, there is considerable variability in the latency due to interference from other users. While low latency 5G links will become available in the near future, the low cost of WiFi associated with operation in unlicensed bands and ease of setup, will continue to make WiFi an attractive technology for many Edge computing applications. Another source of of latency is due to head-of-line blocking from bufferbloat associated with transmit buffering of feature vectors and key-frames. In a video stream with low scene dynamics (key-frames and feature vectors with low temporal variation), a new feature vector (key-frame) can be discarded if it is similar to a previously transmitted feature vector (key-frame), leading to low transmit buffer ingress rate. Conversely, high scene dynamics results in minimal discarding of feature vectors (key-frames), and consequently, a high transmit buffer ingress rate. Additionally, if the wireless network is congested due to interference in the channel, the egress rates from the transmit buffer is low, leading to transmit buffer bloat.

Other sources of latencies such as read/write operations involving persistent storage such as flash (hundreds of microseconds) are far less significant as compared to that due to the wireless link, and transmit buffer bloat. Additionally, prior work has made available low latency data stores (for example, RAMCloud[18], RocksDB[3]) that utilize RAM-based log structured data structures to effectively mask latencies associated with persistent storage.

Based on the above observations on latency sources, our goal is to design latency tuning knobs that a controller can use to tune the latency associated with channel interference, and transmit buffer bloat. A simple solution of using static priority classes per traffic has the disadvantage of prioritizing traffic even under conditions where the channel is not congested. Intermittent congestion is dynamic,

and effectively addressing it requires a dynamic controller. Our key idea is to exploit the differing latency requirements of feature vectors, and key-frames, and the possibility of tolerating loss of key-frame accuracy. The first control knob, *key-frame TX*, determines the rate at which the key-frame is transmitted. When channel interference is detected, the key-frame transmission rate is reduced by the *key-frame TX* knob so as to improve the signal-to-noise ratio (SNR), and thereby the feature vector packet delivery probability (and hence latency) is improved. Unfortunately, reducing the transmission rate of key-frames can exacerbate the bufferbloat problem. To ameliorate bufferbloat, a second control knob, *key-frame Sim*, that determines the degree of similarity of key-frames in the buffer, is introduced. When transmit bufferbloat exceeds a high-threshold, the *key-frame Sim* knob discards K closest matching key-frames from the buffer, thereby reducing the head-of-line blocking experienced by key-frames. The discarding of the key-frames can be considered as trading off accuracy for latency. A controller located at the End-node utilizes the two control knobs and a suitable control policy to ensure the best possible latency for the latency critical feature vectors under varying operating conditions (channel interference, and video scene dynamics).

5 Data store design

The key-frames and feature vectors obtained from the video processing engines at the End-nodes are time-stamped and inserted along with the unique node ID into the respective transmit buffers to be transferred to the Edge-node server. An image similarity index such as Structural Similarity (SSIM) [27] is used to discard temporally adjacent images that are similar above a threshold level.

The Edge-node server implements a persistent low-latency data store such as RocksDB [3] or a distributed low-latency data store such as RAMCloud [18] depending on the data size and fault tolerance to node failure requirements. Note that we avoid persistent storage at the End-node, since the End-node could be illegally accessed because of limited physical security. Encrypted persistent storage may be used at the Edge-nodes; however, this entails additional encryption latency. The Edge-nodes backup the data to a Cloud storage; this operation is considered to be latency insensitive.

6 Evaluation and Results

Our primary goal was to prototype an emulation testbed that can be used to explore various aspects of the Edge data store for distributed machine vision including video workloads, wireless channel interference, data structures, key-frame similarity, and control algorithms. We use LXC containers to emulate End and Edge nodes. Containers implement light-weight OS level virtualization allowing a large number of containers to be spun-up on a single physical machine. The one-hop WiFi network was simulated using an 802.11a based adhoc WiFi network model simulated using the NS3 [2] network simulator. The LXC containers emulating the nodes are connected to NS3 using a tap-bridge device allowing communication between the nodes through the simulated WiFi network. The End-node and Edge-node servers were implemented in Golang and support an RPC interface. Image similarity was computed using the Python sckit-image[5]. BadgerDB [1], a Golang implementation of RocksDB, was used to implement the persistent key-value store at the Edge-node.

We use a synthetic workload consisting of key-frames with an average size of 500 KB, and feature vectors of size 4 KB. High scene (low scene) dynamics was simulated by deriving the key-frames from a distribution with 0.1 (0.9) probability of temporally adjacent key-frames being sufficiently similar to allow discard. The wireless channel interference was modeled through a 1 second long transmission generated by a Poisson process with a mean arrival time of 30 seconds. The TCP latency was measured using the Linux *qperf* utility. Figure 2 shows the latency CDF with and without latency control for high scene dynamics using the *key-frame TX* control knob described in Section 4. A simple bang-bang controller was used - the key-frame transfer from the End-node to the Edge-node was disabled if the latency mea-

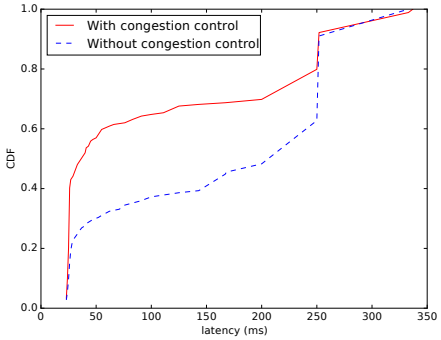

Figure 2: Feature vector latency CDF at high scene dynamics and intermittent channel interference

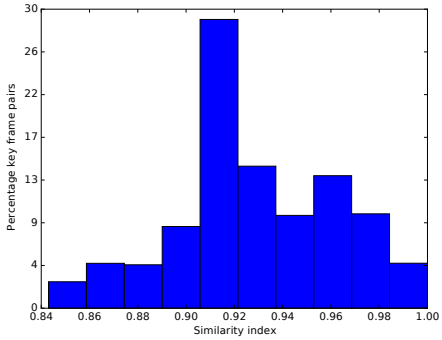

Figure 4: Image similarity index histogram of a pedestrian car accident surveillance video

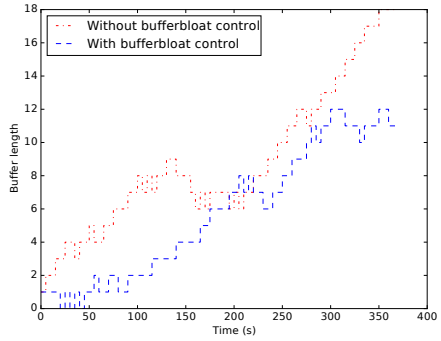

Figure 3: Transmit buffer queue length at high scene dynamics and intermittent channel interference

sured with *qperf* exceeded 25 ms. As seen from Figure 2, the controller improves the feature vector latency CDF, with the median latency improving by 84.5%. However, the latency at the tail is determined by the limited bandwidth of the simulated wireless channel (200 KB/s). Figure 3 shows the resulting increasing bufferbloat of the transmit buffer queue. The bufferbloat was controlled using a simple bang-bang controller using the *key-frame Sim* control knob - if the buffer length exceeded 12 key-frames, K = 2 key-frames was randomly chosen and dropped from the transmit queue. Note that the choice of parameters (for example, buffer length and latency thresholds), are determined by the application. Additionally, the control architecture is scalable since the individual End-node controllers operate independently.

To understand the scene dynamics of real-life Edge computing video workloads, we studied a pedestrian car accident surveillance video obtained from YouTube. Figure 4 plots the histogram of the image similarity of all keyframe pairs in the video. We note that a policy of randomly dropping keyframes yields 63.1% less dissimilar keyframe pairs in the top 20 percentile as compared to the computationally expensive exhaustive evaluation of SSIM index of all image pairs.

7 Conclusions and Future work

In this paper, we have laid the ground work for a data store architecture for latency critical distributed vision applications at the Edge that incorporates application specific design, and autonomous computing techniques.

Future extensions of our work include - node scalability studies, characterization of scene dynamics from multiple surveillance video benchmarks, incorporation of measurement based WiFi channel interference [15], and investigation of more sophisticated control algorithms as described in [19]. An experimental testbed would serve to evaluate the proposed approaches under real-life conditions.

References

[1] Fast key-value db in Go. `https://blog.dgraph.io/post/badger/`. Accessed: 2018-03-27.

[2] NS-3. `https://www.nsnam.org/`. Accessed: 2017-03-03.

[3] A persistent key-value store for fast storage environments. `https://www.rocksdb.org/`. Accessed: 2018-03-27.

[4] Redis. `https://redis.io/`. Accessed: 2017-11-14.

[5] Scikit-image - Image processing in Python. http://scikit-image.org/docs/dev/api/skimage.measure.html.

[6] Baker, J., Bond, C., Corbett, J. C., Furman, J., Khorlin, A., Larson, J., Leon, J.-M., Li, Y., Lloyd, A., and Yushprakh, V. Megastore: Providing scalable, highly available storage for interactive services.

[7] Bonomi, F., Milito, R., Natarajan, P., and Zhu, J. *Fog Computing: A Platform for Internet of Things and Analytics*. Springer International Publishing, Cham, 2014, pp. 169–186.

[8] Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing* (New York, NY, USA, 2012), MCC '12, ACM, pp. 13–16.

[9] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. Dynamo: Amazon's highly available key-value store. *ACM SIGOPS Operating Systems Review 41*, 6 (2007), 205–220.

[10] Drolia, U., Guo, K., Tan, J., Gandhi, R., and Narasimhan, P. Cachier: Edge-caching for recognition applications. In *37th IEEE International Conference on Distributed Computing Systems, ICDCS 2017, Atlanta, GA, USA, June 5-8, 2017* (2017), pp. 276–286.

[11] Hiland-Jrgensen, T., Kazio, M., Poland, T., Taht, D., Hurtig, P., and Brunstorm, A. Ending the anomaly: Achieving low latency and airtime fairness in wifi. In *Proceedings of the 2017 USENIX Annual Technical Conference* (2017), USENIX ATC 17, USENIX, pp. 139–151.

[12] Lakshman, A., and Malik, P. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review 44*, 2 (2010), 35–40.

[13] Lecun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature 521*, 7553 (5 2015), 436–444.

[14] Lee, E. A., Hartmann, B., Kubiatowicz, J., Rosing, T. S., Wawrzynek, J., Wessel, D., Rabaey, J., Pister, K., Sangiovanni-Vincentelli, A., Seshia, S. A., Blaauw, D., Dutta, P., Fu, K., Guestrin, C., Taskar, B., Jafari, R., Jones, D., Kumar, V., Mangharam, R., Pappas, G. J., Murray, R. M., and Rowe, A. The swarm at the edge of the cloud. *IEEE Design Test 31*, 3 (June 2014), 8–20.

[15] Lee, H., Cerpa, A., and Levis, P. Improving wireless simulation through noise modeling. In *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on* (2007), IEEE, pp. 21–30.

[16] Lu, Y., Chowdhery, A., and Kandula, S. Visflow: A relational platform for efficient large-scale video analytics. Tech. rep., June 2016.

[17] Mortazavi, S. H., Salehe, M., Gomes, C. S., Phillips, C., and de Lara, E. Cloudpath: A multi-tier cloud computing framework. In *Proceedings of the 2nd Annual International Symposium on Edge Computing* (New York, NY, USA, 2017), SEC '17, ACM.

[18] Ousterhout, J., Gopalan, A., Gupta, A., Kejriwal, A., Lee, C., Montazeri, B., Ongaro, D., Park, S. J., Qin, H., Rosenblum, M., Rumble, S., Stutsman, R., and Yang, S. The ramcloud storage system. *ACM Trans. Comput. Syst. 33*, 3 (Aug. 2015), 7:1–7:55.

[19] Patikirikorala, T., Colman, A., Han, J., and Wang, L. A systematic survey on the design of self-adaptive software systems using control engineering approaches. In *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems* (Piscataway, NJ, USA, 2012), SEAMS '12, IEEE Press, pp. 33–42.

[20] Sabella, D., Vaillant, A., Kuure, P., Rauschenbach, U., and Giust, F. Mobile-edge computing architecture: The role of mec in the internet of things. *IEEE Consumer Electronics Magazine 5*, 4 (Oct 2016), 84–91.

[21] Sapienza, M., Guardo, E., Cavallo, M., Torre, G. L., Leombruno, G., and Tomarchio, O. Solving critical events through mobile edge computing: An approach for smart cities. In *2016 IEEE International Conference on Smart Computing (SMARTCOMP)* (May 2016), pp. 1–5.

[22] Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing 8*, 4 (Oct 2009), 14–23.

[23] Shi, W., Cao, J., Zhang, Q., Li, Y., and Xu, L. Edge computing: Vision and challenges. *IEEE Internet of Things Journal 3*, 5 (Oct 2016), 637–646.

[24] Truong, B. T., and Venkatesh, S. Video abstraction: A systematic review and classification. *ACM transactions on multimedia computing, communications, and applications (TOMM) 3*, 1 (2007), 3.

[25] Vermesan, O., Friess, P., Guillemin, P., and Gusmeroli, S. *Internet of Things Strategic Research Agenda*. River Publishers, 2011.

[26] Wang, X. Intelligent multi-camera video surveillance: A review. *Pattern Recogn. Lett. 34*, 1 (Jan. 2013), 3–19.

[27] Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. Image quality assessment: From error visibility to structural similarity. *Trans. Img. Proc. 13*, 4 (Apr. 2004), 600–612.