Cascade: An Edge Computing Platform for Real-time Machine Intelligence (Invited paper)

WEIJIA SONG, YUTING YANG, THOMPSON LIU, ANDREA MERLINA*, THIAGO GARRETT*, ROMAN VITENBERG*, LORENZO ROSA*, AAHIL AWATRAMANI, ZHENG WANG, KEN BIRMAN,

Dept of Computer Science, Cornell University, USA

1 INTELLIGENCE FOR THE IOT EDGE

Intelligent IoT is a prerequisite for societal priorities such as a smart power grid, smart urban infrastructures and smart highways. These applications bring requirements such as real-time guarantees, data and action consistency, fault-tolerance, high availability, temporal data indexing, scalability, and even self-organization and self-stabilization. Existing platforms are oriented towards asynchronous, out of band upload of data to the cloud: Important functionality, but not enough to address the need. Cornell's Cascade project seeks to close the gap by creating a new platform for hosting ML and AI, optimized to achieve sharply lower delay and substantially higher bandwidth than in any existing platform. At the same time, Cascade introduces much stronger guarantees - a mix that we believe will be particularly appealing in applications where events should trigger a quick and trustworthy response. This short paper is intended as a brief overview of the effort, with details to be published elsewhere.

2 IOT IS FINALLY A PRACTICAL REALITY

After decades of promise intelligent edge IoT might finally be practical. Three developments stand out:

(1) *Cellular 5G.* Higher speeds and universal connectivity will allow us to connect the cloud to a wider range of sensors and actuators, while novel capabilities such as 5G "slicing" should facilitate predictable latency and bandwidth for critical I/O paths.
(2) *Cloudlets.* Edge networking and content-caching providers are building mini-datacenters (small clusters). These hotspots can host applications, just like the normal cloud - enabling a new edge cloud. Time-sensitive networking protocols and real-time data distribution standards should quickly emerge as these cloudlets are deployed.
(3) *Collaborative ML.* There are many situations in which two or more ML subsystems are combined: the output of one becomes input to the next, yielding a graph of heterogeneous, collaborative MLs. As developers create hybrid solutions, they will require runtime infrastructures for executing intelligent logic under timing pressure.

The kinds of settings that motivate our work include intelligent roadways. Today's vehicles already have automated tracking of lane positioning and the ability to warn of sudden dangers, but lack active help. What if the the road itself could inform the vehicle of hazards? If traffic intersections could transmit warnings (Fig. 1), cars could brake when a cyclist maneuvers dangerously from a hidden location. A smart roadway could go even further, warning cars not to change lanes if a motorcycle is approaching in the interlane space (Fig. 2).

Breaking these tasks down, one would find a number of sub-problems, each of which might require a distinct intelligent subsystem. First, using cameras, lidars and other sensors, we will need to capture images showing the overall state of some portion of the roadway. These may result in large objects (tens of megabytes or more), and the ML models

Contact authors: Weijia Song and Ken Birman, Dept. of Computer Science, Cornell University, Ithaca NY. ws393@cornell.edu, kpb3@cornell.edu *Rosa has a primary affiliation at the University of Bologna, Italy. Vitenberg, Merlina and Garrett have a primarily affiliation at the Univ. of Oslo in Norway.

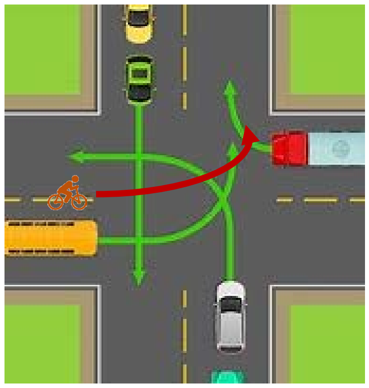
Fig. 1. Smart traffic intersection

Fig. 2. Motorcycle passing between lanes of traffic

used to process them can also be large (hundreds of megabytes or gigabytes). By segmenting and classifying objects present in the images, the system can categorize the type of vehicles, which might trigger a second stage analysis specific to the vehicle class, perhaps to predict trajectories. Recent history might be useful: a motorcycle that has been weaving through traffic and varying its speed is somewhat likely to continue to do so; one that has maintained its path and speed for a while will probably continue to drive in that same style. Intent of the surrounding cars might also enter into decision-making: a car signaling left won't always make a left turn (or shift to the next lane to the left), but that observation unquestionably elevates probabilities. A smart car with an active route recommendation system could even inform the roadway or intersection ahead of time: "this vehicle is likely to exit at the next intersection." Central to our work is the premise that such systems will require collaboration between collections of ML subsystems[1].

Collaborative ML isn't inevitable: one can certainly imagine a single ML trained to operate an entire intersection, receiving a set of inputs that represent the full sensor data set at some instant, and then generating a safety indicator or a warning. But as we scale to more and more complex scenarios, such as the smart roadway, we quickly see that there will need to be a number of distinct specialists. Some MLs would track trajectories of each moving entity and predict intent. Others might assess visibility conditions, and still others report on known hazards, such as potholes or road repair equipment. We can recognize this as a dimension of scalability: rather than one ML that scales to deal with more and more moving parts and more and more hazards and challenges, we instead scale by having more and more expert subsystems, each attentive to a distinct element. Collaborative heterogeneous ML will be obligatory in such settings.

The technology enablers for collaboration are familiar: the output of one program (a machine intelligence) becomes input to the next, yielding a graphical structure in which information flows from sensors through a network of MLs. The needed functionality can be supported by shared files or databases (one stage writes the file, the next one consumes it), pipes, or key-value storage. The actual data shared would be specific to the machine intelligence systems: they could be files containing images or video snippets, structured tables, or even numerical objects such as tensors.

Our work is indifferent to this level of detail – Cascade adopts a key-value perspective but uses file names as keys and offers a FUSE-based POSIX file system API as a secondary option, aiming for something as universal as possible, but also as efficient as possible (with files, multiple system calls are needed to create, update or read data; with a key-value

[1]Readers may be more familiar with *federated* ML, in which a single ML is trained on distinct shards of data, or distributed AI, in which one AI algorithm runs on a cluster of machines. Our work can support those patterns too, but we believe that *collaborative* ML is more descriptive of edge intelligence.

system, the entire object can be created, read, or replaced in one operation). This very general approach can support programs written in any of the popular frameworks (PyTorch, Tensor Flow, Spark, Julia, MXNet, etc.)

We are already learning of many such problems. Companies that sell satellite images must control imaging objectives, degree of zoom and depth of field desired, spectral settings, etc. Some customers might reject photos obstructed by smoke or cloud cover, while others (perhaps, a customer tracking deforestation) might explicitly seek photos showing smoke or dust. To maximize revenue, an adaptive, intelligent solution is needed. Drone control for "last 100m" product deliveries demands real-time edge computing. The same is true for many 5G product ideas, such as augmented VR, remote telepresense and collaborative gaming.

Similar opportunities exist across the spectrum: smart homes that assist with independent living for residents who might otherwise need managed care. A smarter power grid that coordinates power delivery with the community consuming the power would be able to smooth demand and make better use of renewables. Better medical care could assist individuals with chronic problems, particularly in their daily lives outside the medical office. Smart water delivery systems for cities could sense and localize leaks to reduce water waste.

Moreover, although our work is motivated by the need for rapid responsiveness in these kinds of IoT edge settings, there are also many "standard" cloud-computing services in which machine intelligence is being deployed under conditions demanding quick responses. Data centers are evolving to use ML in routing, dynamic resource provisioning for elasticity and DDoS protection. We mentioned cellular 5G earlier: in such systems, mobility predictions are key to uninterrupted service (as customers move from cell to cell, the provider must migrate functionality without visible disruptions such as connectivity loss). Factories are increasingly automated, and integrated with cloud-hosted control infrastructures. Data center firewalls, NAT boxes and routers are increasingly controlled by intelligent systems that dynamically optimize routing and balance loads even as they sense and repel attacks.

3 CASCADE

In support of these kinds of applications, Cornell is creating a new edge computing platform for real-time ML. Cascade:

(1) *Leverages RDMA and GPU accelerators.* Cascade is a key-value store (KVS) built on Derecho, a library supporting ultra-fast RDMA multicast [4, 11, 12, 16]. This multicast comes in two forms: atomic multicast for data held in memory, and Paxos for durable versioned data (append-only logs). A novel stability barrier ensures that queries only see durable, strongly consistent data: a form of snapshot isolation [11]. Should a failure disable the system, Cascade will self-repair, then resume normal operations.

(2) *Achieves sharply lower delays and very high bandwidth.* In our early experiments, Cascade achieves delays as low as 12*us* for triggering events, with bandwidth exceeding 8GB/s: 100 to 10,000 times faster than popular event-processing frameworks. Although these measurements were on 100Gbps RDMA, Cascade can also be used over TCP, and work is underway to support a new industry standard for user-space networking called the Dataplane Developers Kit, DPDK.

(3) *Extensible with user-supplied logic.* Unusually, Cascade hosts computation within the key-value store to minimize delays when data is collocated with the computation that needs to access it. We call this the Cascade fast-path, and it centers on logic dynamically loaded into the address space of the KVS servers themselves, where they can be executed when an event of interest occurs (similar to stored procedures in a database). Jointly, these mechanisms enable pipelines of actions with extremely low latency within which tasks will execute instantly when the corresponding event fires.

(4) *Intelligent caching.* Sometimes data must be downloaded from databases or other sources, by issuing a query (often from within the programming language: in all the popular ML platforms, embedded queries are a common option). Here, there may be value in caching so that these downloads won't be unnecessarily reissued if a similar event triggers shortly after the first one. Caching on the GPU accelerator can be valuable too: Often, ML programs would require objects holding their hyperparameters and the pretrained model, and these could be huge (hundreds of megabytes or gigabytes). Cascade addresses both needs.

(5) *Lock free updates.* Because Cascade uses atomic multicast and Paxos when updating key-value data within a shard, updates can be seen as a form of one-shot transactions. There is no way to perform a full transaction on data scattered over multiple shards, to avoid the huge (albeit polynomial) overheads noted in Gray's famous paper on naive database replication costs [10]. On the positive side, though, no locks are needed for these one-shot transactions, because our multicast delivery layer is single threaded. As mentioned earlier, queries benefit from a form of fully serializable snapshot isolation: we permit access to key-value tuples only after the multicast stabilizes, which normally takes no more than 15us. The effect is that any query runs on a stable prefix - in effect, a write-once, read-many log. By building appropriate indexing structures, we can support queries that extract entire tensors in a single request (perhaps, a time axis, and then a series of n-dimensional "slices" across some data set at each instant in time). The interface is exceptionally performant and easy to use.

As a very simple illustration of these ideas, we used the Cascade prototype to create an AI-assisted dairy farm infrastructure, one element of which is illustrated in Fig. 3. Obviously, a smart dairy isn't remotely as delay-sensitive as the examples we gave earlier: we fully intend to tackle them, but are trying to get there in a step by step manner. The most demanding step in this application is shown in the figure, and entails an interaction that arises when uploading and analyzing an image. It involves deciding whether the image properly frames the animal of interest (using an exchange of thumbnails and other image meta-data to optimize bandwidth), verifying that there are no humans visible (a farm-workers union requirement), then uploading a high-resolution version directly to GPU for detailed analysis. Decisions must be made within 25-50ms, before the cow exits the passage where the camera is deployed. The analytic function itself resides in the Cascade server, side by side with the objects it uses and the accelerators that run any compute-intensive ML kernels. This entire structure is then scaled horizontally, yielding a multinode service that is sharded, with each shard replicated for high availability.

Notice that the figure illustrates two upload paths, and that there is a dashed download line from the Cascade storage layer to the GPU. These illustrate aspects of the system that turn out to be important for performance. Several factors come into play. First, consider the user code for processing this stage of the computation, which will be watching for events of interest (technically, monitoring certain keys or folders, watching for an upload that should trigger computation). If we deploy this user code in a separate address space, some form of copying or memory-mapped sharing must occur to avoid copying from address space to address space. Our design opts for a "both" approach: we do support applications that run external to Cascade, via a standard RPC interface (accelerated by RDMA if the application is on a separate computer and RDMA is available), but our fast path hosts the ML code directly in the Cascade address space, avoiding copying for user code that employs the corresponding API. We are not aware of prior systems that compute, store data and host user code all in one address space, but for our low-latency goals, doing so is extremely valuable.

Next, recognizing that many ML applications use GPU or other accelerators, we need to envision cases in which the user code would promptly invoke an accelerator-hosted kernel, such as a matrix multiply. It would be inefficient to send large objects to the user logic (the lambda) only to have the user's code turn around and copy them to GPU

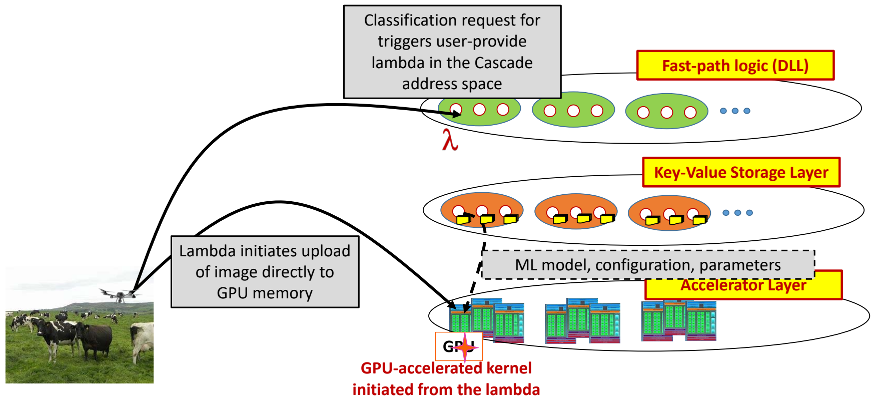
Classification request for triggers user-provide lambda in the Cascade address space
Fast-path logic (DLL)
λ
Key-Value Storage Layer
ML model, configuration, parameters
Accelerator Layer
Lambda initiates upload of image directly to GPU memory
GPU
GPU-accelerated kernel initiated from the lambda

Fig. 3. Cascade-Based Service With User Logic

memory. Accordingly, we would ideally want the inputs to be cached on GPU or at least preloaded as early as possible: the dashed line. Indeed, because RDMA has begun to support direct transfers from one machine to GPU memory on a second machine, we would ideally leverage that option to send our image data to the GPU, where computation can instantly be launched.

Finally, there is a non-trivial scheduling challenge. When objects are stored in Cascade, it is important to group them so that all the inputs to each lambda are available in the same shard, and then to route the computational trigger event to some member in that shard. This scheduler must load-balance but also keep the hardware resources usefully busy. TIDE, the Cascade scheduler, plays these roles (and additionally, manages the host and GPU caches).

4 RELATED WORK

Today most edge-computing systems are simply scripted Linux applications that form a pipeline in which data is shared through files. However, there are some tools, such as Apache Flink [5], for event stream processing (often as a first stage in front of Spark or Tensor Flow). Most such tools, including Flink, focus on batching: they are very inefficient in an event-by-event time-pressured situation, and also at risk of user-visible data inconsistencies arising from the underlying HDFS file system [15]. Yet batching introduces high latency. In contrast, Cascade was created from the ground up for event-driven settings where timely response is prioritized. Additionally, the style of ML federation on which our work focuses have not been a primary emphasis in prior work.

Also relevant are the model-serving IoT edge infrastructures, such as TensorFlow Serving [14], NVIDIA Triton [2], and TorchServe [1]. As the names suggest, these are applications that can host collections of ML models, optimized to serve client requests that specify the desired model and provide required inputs. Cascade could support model servers, and they should gain substantial efficiencies by running on our framework. Our preliminary review suggests that these three model servers could be recompiled as a DLL and then loaded dynamically for use on the Cascade fast path, but additional experimentation is needed.

Cascade is a KVS. Prior work on KVS platforms includes RDMA-enabled systems such as FARM [7] and FASST [13] as well as commercial products, such as Amazon's DynamoDB [6], Snowflake [3], Microsoft Cosmos [8, 9], Databricks Datalake, Cassandra, RocksDB or even the Ceph object-oriented file system, which runs over a key-value store called

RADOS [17]. The distinction here is that none of these solutions host the developer-supplied lambdas within the same address space as the storage system and the hosted GPU accelerator, forcing costly locking, copying and domain crossings. Our contribution centers on the Cascade fast-path, which hosts lambdas in the address space of Cascade itself, as well as our use of RDMA for lock-free and zero-copy hardware accelerated data movement. This permits multiple orders of magnitude improvements both in event-response latency and in overall throughput.

5 CONCLUSION

Cascade enables low-latency ML-assisted analysis of images and other kinds of (potentially large, high-rate) data, while also providing guarantees of data consistency, high availability, self-repair after failures and durability for updates. The system is compatible with popular platforms while also offering a more customized option for user-supplied extensions that achieves the highest possible speed using a lock-free, zero-copy data path.

6 ACKNOWLEDGEMENTS AND DOWNLOAD SITE

This work was supported by grants from AFRL/RY SWEC, The Norwegian Science Foundation, Microsoft and Siemens. Cascade is open source and available from https://github.com/Derecho-Project/Cascade.

REFERENCES

[1] Torchserve. https://github.com/pytorch/serve.

[2] Triton inference server. https://github.com/triton-inference-server/server.

[3] Snowflake. https://snowflake.com, 2012.

[4] Jonathan Behrens, Sagar Jha, Ken Birman, and Edward Tremel. Rdmc: A reliable rdma multicast for large objects. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 71–82. IEEE, 2018.

[5] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015.

[6] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's Highly Available Key-value Store. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 205–220, New York, NY, USA, 2007. ACM.

[7] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. Farm: Fast remote memory. In *11th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)*, pages 401–414, 2014.

[8] Seth Elliot. Microsoft Cosmos: Petabytes perfectly processed perfunctorily. https://blogs.msdn.microsoft.com/seliot/2010/11/05/microsoft-cosmos-petabytes-perfectly-processed-perfunctorily/, Nov. 2015.

[9] Mary Jo Foley for All About Microsoft. The Bing back-end: More on Cosmos, Tiger and Scope. http://www.zdnet.com/article/the-bing-back-end-more-on-cosmos-tiger-and-scope/, Oct. 2011.

[10] Jim Gray, Pat Helland, Patrick O'Neil, and Dennis Shasha. The dangers of replication and a solution. *ACM SIGMOD*, 25:173–182, 1996.

[11] Sagar Jha, Jonathan Behrens, Theo Gkountouvas, Matthew Milano, Weijia Song, Edward Tremel, Robbert Van Renesse, Sydney Zink, and Kenneth P Birman. Derecho: Fast state machine replication for cloud services. *ACM Transactions on Computer Systems (TOCS)*, 36(2):1–49, 2019.

[12] Sagar Jha, Lorenzo Rosa, and Ken Birman. Spindle: Techniques for optimizing atomic multicast on rdma. In *2022 IEEE 42st International Conference on Distributed Computing Systems (ICDCS)*, 2022.

[13] Anuj Kalia, Michael Kaminsky, and David G Andersen. Fasst: Fast, scalable and simple distributed transactions with two-sided ({RDMA}) datagram rpcs. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 185–201, 2016.

[14] Christopher Olston, Noah Fiedel, Kiril Gorovoy, Jeremiah Harmsen, Li Lao, Fangwei Li, Vinu Rajashekhar, Sukriti Ramesh, and Jordan Soyke. Tensorflow-serving: Flexible, high-performance ml serving. *arXiv preprint arXiv:1712.06139*, 2017.

[15] Weijia Song, Theo Gkountouvas, Ken Birman, Qi Chen, and Zhen Xiao. The freeze-frame file system. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, SoCC '16, page 307–320, New York, NY, USA, 2016. Association for Computing Machinery.

[16] Edward Tremel, Sagar Jha, Weijia Song, David Chu, and Ken Birman. Reliable, efficient recovery for complex services with replicated subsystems. In *DSN*, pages 172–183. IEEE, 2020.

[17] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 307–320, Berkeley, CA, USA, 2006. USENIX Association.