CASCADE: ULTRA-FAST EDGE COMPUTING FOR INTELLIGENT IoT

Ken Birman
Cornell University
THE WORLD IS GENERATING A NEW WAVE OF IOT/ML PIPELINES... THERE ARE MANY USE CASES

Satellite Images  AI/ML Layer  Smart Queries

“Which farms will be at risk of wheat blight in 2021?”
... ROLLOUT OF 5G IS SPAWNING MANY IOT EDGE OPPORTUNITIES

5G is like gigabit networking to phones, devices and cars – everywhere!

5G mobile users move from base-station to base-station as they bike/drive/fly. Each base station has a compute cluster to support those users.

Role of AI/ML is to enable continuous intelligence in the apps. (This will create a new infrastructure often called edge cloud).
**AI/ML ELEMENTS INTERACT!**

Clearly we will have

- Many intelligent components
- Often one AI/ML component will be dependent on the actions or outputs generated by some other AI/ML component
- All of this in real-time and sometimes, with large objects like photos or video or lidar
CONCEPT: FEDERATED AI/ML

Increasingly seen in robotics, smart homes, 5G, digital twin scenarios.

- The application is a **graphical collection** of AI classifiers / learners
- Nodes represent computational tasks.
- Edges represent data flow between distinct tasks.
- The flow may be implicit – coding style doesn’t require that developers know or create this graph…
DISTRIBUTED AI

In fact even a standard AI algorithm, such as logistic regression, might have many moving parts spread over many machines.

D-AI: an AI algorithm implemented as a distributed program.

Federated ML, distributed AI and edge IoT result in D-AI and D-ML graphs of small components — $\lambda$ functions...
EACH $\lambda$ REPRESENTS A DISTINCT ELEMENT

Lambda notation: shorthand for a “computed function”. Each performs an AI task, such as image classification.

Thus, we build pipelines or directed graphs linking tasks.

Today’s cloud platforms have limited support for this model, lack the real-time and consistency guarantees needed for IoT.
CASCADe GOALS?

Cornell set out to build a runtime system for edge IoT that is

- Really easy to use, like Tensor Flow, Pandas, Spark, …
  and back-compatible: users can import existing code easily
- Efficient, cost-effective, scalable even to global deployments
- Offers real-time responsiveness in addition to consistency
HOW DO AI/ML SYSTEMS SCALE? SHARDING

- Start with a directory structure, like on Windows or Linux
- Take the file system pathname (from the root), hash it to create a pseudo-random 64-bit key. Think of the file as a (key,value) tuple. (Here the “value” is the byte vector of data in the file).
- Spread these tuples evenly over S servers.
- A sharded storage system would often be referred to as a key-value store (KVS) or distributed hash table (DHT)

Nine (or more) service members organized into three shards, each with three members

KEN BIRMAN (KEN@CS.CORNELL.EDU)
SHARDING IS AUTOMATIC AND INVISIBLE!

Your data is in this storage layer!

User thinks of the data as being in one big file system or database

… in practice the data is split into chunks and then replicated for fault-tolerance, all automatically and transparent to the user

Actual Storage Layer

Perceived Storage Layer
We could focus on speeding up interactions with sharded services.

... For example, by using RDMA to accelerate data movement and replication (used by sharded services for fault-tolerance).

RDMA offloads TCP into network hardware for exceptional speed.

This will be faster, but we still end up moving data often.
IS DATA MOVEMENT A HIGH OVERHEAD?

Yes. With modern ML you need to think about

- The image or video we are trying to understand
- The ML model used to understand it (like a CNN)
- The model hyperparameters (configuration) and model data

These objects can be hundreds of megabytes in size, and even with RDMA it takes too long to move them.
SO OUR “SHARDED CHALLENGE” IS...

To scale, we need sharding. This is unavoidable and universal.

Minimizing data movement implies a need to preplan, so that our ML model and its dependent data are prepositioned in the ideal place. As soon as our acquired image gets there, the ML react “instantly”.

This activity will occur in edge computing clusters — perhaps on the 5G PoP sites, close to the mobile user.
Prepositioned code and data

What if we load application logic directly into the same address space as the storage solution?

Now the application can find data “locally” and we eliminate many data fetch delays!

This is how Cascade works. In fact the code is in the Cascade address space itself, much like loading a DLL in Linux.
We run Cascade on a set of nodes. Here we see nine nodes in three shards.

A shard identically replicates (key,value) tuples, using Paxos.

Here, an ML model for key “Cow Health” was stored in shard 0. “Animal Identification” ended up in shard 2.
Using a Smart Service

This illustrates the idea of data being “ideally” positioned or sent directly to the perfect place...

Skip this transfer for any ML objects already cached in the GPU device!

RDMA directly into GPU memory

Image to classify

Request for classification runs as a lambda

GPU-accelerated kernel initiated from the lambda
**DAIRY IMAGE PIPELINE: FRONT END**

Dairy Farm

Image Pipeline Front End (As an external client)

The Farm Server (IoT Edge)

- Frame Extractor
- Frame Sampler
- Video clip store
- Frame Server

Data Center

Cascade Image pipeline

WAN
Integrate daily data

Upload daily date to Azure Blob Storage

Streaming image frames through TCP portal

Download blobs from Azure & Store to Cascade VCSS subgroup

Filtered image frames

Farm server

External Client to Cascade

LINQ query to retrieve data of most recent 10 days from Cascade about cow

Probability of calving in next 8h is: 80%

ML Model

birth prediction

CV model

Image analysis

cow id: 127


deleted_image_frames

<field>/<cow_id>{<ts(ver)>}
daily_protein/cow_id1{ver_1} = 2.89

daily_protein/cow_id237{ver_38} = 4.42

client = cascade_py.ServiceClientAPI()
daily_yield_pool = client.get_object_pool("VolatileCascadeStore","daily_yield")
daily_yield_pool.put(cow_id,val)

daily_fat/cow_id237{ver_38} = 4.42

CV model

Image analysis

cow id: 127

ML Model

birth prediction

Probability of calving in next 8h is: 80%

LINQ query to retrieve data of most recent 10 days from Cascade about cow

client = cascade_py.ServiceClientAPI()
cow_id = "cow_id28"
recent_data = []
for field in ["daily_yield", "daily_fat", "daily_protein"]:
    field_pool = client.get_object_pool("VolatileCascadeStore",field)
    query_result = field_pool
        .where(lambda x:x.endingwith("/"+cow_id))
        .order_by(lambda x:x.version)
        .reverse()
        .take(10)
recent_data.append(query_result)
PERFORMANCE CHALLENGE

Each small black “code box” is an example ML \( \lambda \), in a pipeline

How **fast** can these triggered actions be initiated? How close to perfect wire efficiencies and **zero extra delay** can we get?
A VERY HIGH-LEVEL WAY TO CREATE AI!

Easy to use... the developer only needs to provide tiny little plug-in functions (the ML $\lambda$ methods)

We saw those in small black boxes

```python
client = cascade_py.ServiceClientAPI()
daily_yield_pool = client.get_object_pool("VolatileCascadeStore","daily_yield")
daily_yield_pool.put(cow_id,val)
```
EVEN COMPLEX IDEAS STILL USE JUST SMALL LAMBDAS. THEY LOOK LIKE DATABASE CODE

```python
client = cascade_py.ServiceClientAPI()
cow_id = "cow_id128"
recent_data = []
for field in ["daily_yeild", "daily_fat", "daily_protein"]:
    field_pool = client.get_object_pool("VolatileCascadeStore", field)
    query_result = field_pool \
        .where(lambda x:x.endswith("/"+cow_id)) \
        .order_by(lambda x:x.version) \
        .reverse() \
        .take(10)
recent_data.append(query_result)
```
IS THIS CODE OR A DATABASE QUERY?

Both!

When working with Cascade, the developer writes code in Python, Java or C++ (C++ is the “fastest”) but can embed SQL queries right into their code, and can call into ML kernels that are preloaded to a GPU or FPGA accelerator.
**OUR JOB AS THE DESIGNERS OF CASCADE**

Ensure that the D-Al or D-ML code has fresh, correct data

Ensure that the code will run “close” to inputs it depends on, with minimal per-event delay, yet keeping resources busy

Map the data movement and access to zero-copy, lock-free data paths where hardware accelerators can be leveraged
VALUE OF CONSISTENCY

Why does it matter?

KEN BIRMAN (KEN@CS.CORNELL.EDU)
Recall: Cascade has a built-in temporal indexing feature. Suppose our distributed AI is triggered by event $\varepsilon$ at time $\tau$.

We run all the lambdas triggered by $\varepsilon$ along a consistent cut “optimally close” to time $\tau$ (and selected deterministically).

Effect: The lambda won’t see platform-induced inconsistencies.
Standard platforms are inconsistent when used under intense delay pressure (left). In Cascade, $\lambda$s run on consistent cuts. GPS-timestamped sensor data results in clean input to the federated-AI algorithm (in this case, a simple visualization).
CENTRAL CONCEPTUAL CHALLENGE

One event may trigger many lambdas.

These lambdas may need to run on multiple nodes... yet will share the same temporal index ($\tau$ from the trigger event $\varepsilon$).

A Cascade query always sees a “consistent state snapshot.”
A temporal query for time $\tau$ sees a consistent cut at $\tau \pm \delta_{\text{clock}}$.

Queries to unstable data must wait, but updates are stable within 50us.
RAW SPEED

The bottom line
IMPLEMENTATION IS UNDERWAY RIGHT NOW

We have some very preliminary data, but much more coming.

Key-value performance looks at the cost of uploading data into Cascade in its two main modes (in memory or persisted to disk)

Pipeline performance looks at a simple linear D-AI/D-ML graph
CASCADE K/V PUT THROUGHPUT

(a) Put with 1MB payload

(b) Put with 10K payload
CASCADE K/V PUT LATENCY (1MB PAYLOAD)

(a) Trigger put

(b) Volatile put

(c) Persistent put
CASCADE K/V STORE LATENCY (10KB PAYLOAD)

(a) Trigger put

(b) Volatile put

(c) Persistent put
CASCADE K/V PUT LATENCY BREAKDOWN

(a) Volatile put
(b) Persistent put
CASCADE PIPELINE THROUGHPUT (TRIGGER PUT)
CASCADE PIPELINE LATENCY (TRIGGER PUT)
QUICK COMPARISON: SPARK ON SAME NODES

<table>
<thead>
<tr>
<th>Pipeline Length</th>
<th>Message Size</th>
<th>Average Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1_10kb</td>
<td>650.76</td>
</tr>
<tr>
<td>1</td>
<td>1_1mb</td>
<td>881.66</td>
</tr>
<tr>
<td>2</td>
<td>2_10kb</td>
<td>986.76</td>
</tr>
<tr>
<td>3</td>
<td>2_1mb</td>
<td>1520.16</td>
</tr>
</tbody>
</table>

... Cascade is definitely faster, and we are not nearly done tuning
HOW FAST IS CASCADE?

Detailed benchmarking underway right now. Challenge is to get as close as possible to zero-copy, zero-locking hardware-accelerated critical path

Preliminary results:
- Sub-millisecond delays when events occur, even with photos.
- Micoseconds for events that have really small payloads
- In our testbed, bandwidth reaches 950Gbits/s on RDMA
- Scales to long pipelines or ML graphs
THE CASCADE AND DERECHO TEAM

Weijia Song
Alicia Yang
Ken Birman
Sagar Jha
Lorenzo Rosa
Mae Milano
(faculty at Augusta Univ.)

Andrea Merlina
Roman Vitenberg
Cornell undergrads:
Aahil Awatramani, Ben Posnick, Max Charlamb, Thompson Liu, Archishman Sravankumar, Aaron Weiss, Peter Zheng

Edward Tremel
(post-doc at Berkeley)
### MANY PAPERS... THE ONES THAT ACTUALLY ARE ABOUT CASCADE ARE STILL TO COME!

<table>
<thead>
<tr>
<th>Title</th>
<th>Authors</th>
<th>Details</th>
</tr>
</thead>
</table>
SUMMARY AND CONCLUSION

To run cloud-style AI on IoT data with fast events, we need a new computing platform.

The Cascade key-value framework is optimized for speed, with special support for “event triggered” D-AI applications.

Cascade makes it easy to move existing ML and AI logic from existing big-data platforms (like Databricks, Tensor Flow) to the IoT edge.