THE PROMISE OF EDGE INTELLIGENCE

Can “real world” artificial intelligence be as accurate and as fast as human intelligence?
THE WORLD IS GENERATING A NEW WAVE OF IOT/ML PIPELINES… THERE ARE MANY USE CASES

Data sources

Cooperative ML
Distributed AI

Smart Queries

How much should I budget for raw milk purchases in March for my yoghurt factory?
COOPERATIVE ML?

One ML to rule them all? Not likely!

So we need to think about cooperation between MLs 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.
A SINGLE $\lambda$ COULD BE AN ENTIRE DISTRIBUTED AI, ON A COLLECTION OF MACHINES

Single $\lambda$ may run on a pool of compute nodes.

Today’s cloud platforms have limited support for this model, lack the real-time and consistency guarantees needed for IoT.
THE NEED IS FOR TIME-FOCUSED EDGE INTELLIGENCE

IoT data arrives as a stream.

Edge intelligence must be instantly reactive

- Get power grid data at 250ms intervals
- Format it as 15 x 15 tensor by GPS location with a time axis from 10:00:00 to 10:01:00
- Run ML phase disruption analytic on the tensor

In today’s cloud, AIs fight platform noise

Cascade: Faster yet accurate
HIGH LEVEL CASCADE GOALS

Legacy support: Easy to use with no need to change your code

Much faster than standard platforms: low delay, high bandwidth

Stronger guarantees: Your ML doesn’t fight platform “noise”
BUT HOW WILL WE GUARANTEE CONSISTENCY? HOW CAN IT BE SO FAST?

Let’s start by asking how to make it fast

Then we will see that we already know how to guarantee consistency: be smart about time (and time skew), and use Chandy-Lamport consistent cuts when we hit the limits of clock precision/accuracy!
TRADITIONAL APPROACH

Your code is in its own address space, maybe on a different computer

Accelerator Layer

GPU

Your logic

Run-time copies data into GPU

Request objects one by one

File system or Key-Value Store

Objects copied over datacenter TCP network
TRADITIONAL APPROACH

Your logic

Caching hides these costs, in iterative cases

Accelerator Layer

Your code is in its own address space, maybe on a different computer

File system or Key-Value Store

GPU

Request objects one by one

Objects copied over datacenter TCP network

$$$

lambda$

$$$
RDMA CAN HELP... A LITTLE

Your code is in its own address space, maybe on a different computer

Request objects one by one

Objects copied over datacenter TCP network

File system or Key-Value Store

Runtime still copies data into GPU

Accelerator Layer

$\lambda$

Your logic

GPU
CASCADE CAN BE USED THIS WAY...

Your logic

Your code is in its own address space, maybe on a different computer

Accelerator Layer

GPU

Request objects one by one

Objects copied over datacenter TCP network

RDMA

Cascade runtime environment

Key-Value Storage Layer
... BUT CAN ALSO HOST USER CODE
CASCADE: CUSTOMIZED SMART SERVICES

Request for classification triggers a C++ lambda in the Cascade address space

Ideally, these are cached on GPU

RDMA directly into GPU memory

Image to classify

Fast-path logic (DLL)

Key-Value Storage Layer

ML model, configuration, parameters

Accelerator Layer

GPU-accelerated kernel initiated from the lambda
FAST-PATH PERFORMANCE

A simple federated ML pipeline

Cascade is close to ideal efficiency on our hardware and 100 to 10,000x faster than common options like Apache Flink
THE CASCADE MODEL

Is it a service? A library?
CASCADE IS A SERVICE

Cascade runs on a set of nodes (machines or VMs) where it controls some resources (cores, RDMA interfaces, GPUs/FPGAs, memory).

Users can build applications that access Cascade from “outside”. We call those “external clients”. They would think of Cascade purely as a key-value storage system, accelerated by RDMA.
But the highest speed is achieved by extending the Cascade service by adding logic ($\lambda$s) that will run inside Cascade.

Used this way, Cascade enables creation of a customized service: a “smart” service ideal for your purposes.
RECAP: EXTENSIBILITY (PAAS MODEL)

Cascade is **one** service

But when you supply customization it acts like many specialized services, one per application

So it becomes a platform for new microservices, like these!
THE CASCADE STORAGE MODEL

How should Cascade data be managed?
CASCADE IS A KEY-VALUE STORE

... but it also supports a file system API

... and the key-value API itself has some fancy options, beyond the basic put/get/watch we learned about in lectures 1-4
IN FACT CASCADE’S FILE SYSTEM IS AN EXAMPLE OF EXTENSIBILITY

Cascade’s file system is implemented as a λ:

- Every object has a pathname.
- The file system extension supports normal file operations.
- You can access it just like any file system.

Similar to Ceph file system, yet just a few lines of code to translate user requests (via the FUSE library) into Cascade put/get/watch
Same model used in cloud file systems like HDFS and Ceph.

- Updates are log appends using Paxos. Each object has a log of versions that evolved over time.
- Reads run on the stable prefix of the log.
VERSIONED UPDATES

Each time you write to an object, Cascade creates a new version.

If you read an object, modify it, and then write it back, you can tell Cascade which version you modified. The `put` will double check to be sure that this is still the current version before replacing it, and otherwise returns an error (then you can loop)
VERSIONED/TEMPORAL QUERIES

Accessed via `get`, but specify the version # or time desired

In the volatile case, Cascade only keeps the most recent version. With persistent objects, Cascade keeps a log of past versions.

- By default, applications see the most current version
- Indexed access allows the application to query any version (by version number or time), or fetch any data range.
VERSIONED OBJECTS

Versioning enables intelligence over a sequence of events

- **key**: The object store *always* tracks information on a per-object basis
- **version-number**: Just an integer
- **time**: If the object itself lacks a timestamp, we just use “platform” time.

Now **get** can lookup most current version, or a specific one, even by time. The object store is optimized to leverage non-volatile memory hardware.
THE CASCADE COMPUTE MODEL

Virtual synchrony, Atomic multicast, Paxos, consistent cuts...
THE CENTRAL PUZZLE

The very fastest data paths require compilation, ideally in languages like C++.

But we want Cascade to run as a service, so it would often already be running when a new user comes along and wishes to create and launch some completely new service.

How can we extend a running system? Actually… it isn’t so hard
FIRST QUESTION: WHAT’S IN A \( \lambda \)?

We support many languages. Native APIs are Python with various packages (including LINQ) and C++ with LINQ.

Code is concise – LINQ pioneered a style that mixes “kernel” invocations with embedded SQL. Maps cleanly to GPU, FPGAs.

Cascade manages GPUs and can cache data in GPU memory.
HOW CAN A KEY-VALUE STORE “BE” A CLASSIFIER SERVICE OR AN ANALYTIC SERVICE?

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 object with the key “Flowers” was stored in shard 0. “Vegetables” ended up in shard 2.
First tier: inexpensive computation on meta-data

Key-value object store holds specialized knowledge models for categories (flowers, birds, dogs, trees…)

**Flowers**

- Sunflower $p=0.97$
- Flower $p=0.85$

**Vegetables**

- Zucchini blossom $p=0.04$
- Vegetable $p=0.6$

**IoT Cloud Infrastructure**

Most likely a sunflower!
Suppose a cooperative 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.
Cascade consistent cuts + GPS-timestamped sensor data result in clean input to the D-AI algorithm (in this case, a simple visualization)
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.
A temporal query at time $\tau$ sees a consistent cut at $\tau \pm \delta$.

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

Each Cascade shard has its own Paxos-based log.

Each $\lambda$ is triggered by an upcall from a "watcher" monitoring some key (or pattern).

Even if $\delta$ is small, there is a choice to make: Which versions will be fetched? Cascade accesses data along a Chandy-Lamport Consistent Cut.
GROUPING OBJECTS

Often a lambda will need to access several objects that should ideally all have their own keys, yet you want them grouped on the same shard.

For this, Cascade supports “affinity grouping”. Each object has a second key, used for placement.

Even if A and B have different keys – “names” – they will be stored on the same shard if you assign them the same affinity key.
WHAT DOES A REAL APPLICATION LOOK LIKE TODAY?

Example, courtesy of Weijia, Alicia and Thompson
DAIRY IMAGE PIPELINE: FRONT END

Dairy Farm

The Farm Server (IoT Edge)

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

Data Center

- Image Pipeline Front End (As an external client)

Cascade Image pipeline

WAN
Streaming image frames through TCP portal

Integrate daily data

Upload daily date to Azure Blob Storage

Filtered image frames

Farm server

External Client to Cascade

Download blobs from Azure & Store to Cascade VCSS subgroup

CV model

Image analysis

Catalogue

LINQ query to retrieve data of most recent 10 days from Cascade about cow 128

ML Model

birth prediction

Probability of calving in next 8h is: 80%
**C++ IS SIMILAR (BUT MORE EFFICIENT)**

<table>
<thead>
<tr>
<th>Date</th>
<th>cow_id</th>
<th>daily_yield</th>
<th>daily_fat</th>
<th>...</th>
<th>daily_protein</th>
</tr>
</thead>
<tbody>
<tr>
<td>12/3/20</td>
<td>1</td>
<td>14</td>
<td>3.96</td>
<td>...</td>
<td>2.89</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1/10/21</td>
<td>237</td>
<td>20</td>
<td>4.42</td>
<td>...</td>
<td>4.55</td>
</tr>
</tbody>
</table>

Streaming image frames through TCP portal

Upload daily date to Azure Blob Storage

Integrate daily data

Download blobs from Azure & Store to Cascade VCSS subgroup

Filtered image frames

Farm server

External Client to Cascade

Cascade backend

Filtered image frames

CV model

Store to subgroup VCSS

Trigger image analysis

ML Model

Probability of calving in next 8h is: 80%

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

<table>
<thead>
<tr>
<th>Date</th>
<th>cow_id</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>12/3/20</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>12/3/20</td>
<td>237</td>
<td>...</td>
</tr>
</tbody>
</table>

ServiceClient API capi:

```
uint32_t num_subgroups = capi.template<SubgroupType> get_number_of_subgroups();
auto sg_idx = std::hash<std::string>()(daily_yield) % num_subgroups;
uint32_t num_shards = capi.template<SubgroupType> get_number_of_shards(sg_idx);
capi.template put<SubgroupType>({val, sg_idx, cow_id % num_shards});
```

```
<field>/cow_id{<ts(ver)>}
daily_protein/cow_id1{ver_1} = 2.89
daily_fat/cow_id237{ver_38} = 4.42
```
THE CASCADE AND DERECHO TEAM

Weijia Song
Alicia Yang
Ken Birman
Sagar Jha
Lorenzo Rosa
Mae Milano
(post-doc at Berkeley)
Andrea Merlina
Roman Vitenberg
Cornell undergrads:
Aahil Awatramani, Ben Posnick, Max
Charlamb, Archishman Sravankumar, Aaron
Weiss, Peter Zheng
Edward Tremel
(faculty at Augusta Univ.)
Thompson Liu