Introduction to Edge AI Systems

Lin Wang
Department of Computer Science
VU Amsterdam

@ EDL Winter School
November 24, 2021
About me

I make networked systems easy to use and easy to manage.

Lin Wang
Assistant Professor
VU Amsterdam

I make networked systems easy to use and easy to manage.

Edge AI systems, in-network computing, and cyber-physical systems

Programming models

Resource management
At the end of this lecture, you will be able to

...get an overview of the evolution of edge computing

...learn how to design efficient edge AI systems for video analytics

...craft a vision for general-purpose edge computing
The era of mobile and ubiquitous applications

A world full of smart devices and mobile/ubiquitous applications, powered by the rapid advancements of **computing** (cloud, accelerators), **wireless communication** (5G and beyond, WIFI6, BLE), and **artificial intelligence** (deep learning, reasoning).
Computing demands/requirements

From these mobile/ubiquitous applications

- Much increased **data volume**
  (Video/audio streams, sensor/LiDAR, etc.)

- Much increased **computational complexity**
  (Modern learning applied on big data)

- Bounded **latency** becomes a key concern
  ("Realtime goes Internet!")

- Growing importance of **mission-critical** applications
  (Safety, reliability, and security)
Computing status-quo: two extremes

On-device computing
- Low latency
- No network bottleneck
- Degraded service quality (limited computing power $\rightarrow$ DNN models being compressed, pruned, or quantized)

Cloud computing
- High service quality
- High bandwidth consumption (data streaming)
- High and uncertain latency (best-effort nature of the Internet)
Modern distributed computing continuum

Covering a wide range of tradeoffs between latency and service quality

- **Cloud** (centralized data centers)
- **Edge** (PoP, cellular base stations, APs)
- **Devices** (mobile/wearables, robots, CPS)

Technologies:
- 5G
- WiFi
- V2X
- LoRa
What do we mean by “edge” in this lecture?

**Edge servers / DCs**
(Purposefully deployed computing resources at the edge)

**Network devices**
(Routers/switches that can perform computation)

**Mobile and IoT**
(Devices that are tailored for specific use cases)

We call these the **edge infrastructure**

We call these end-devices
Agenda

Mobile computation offloading
The history and motivation behind edge computing

DL-based edge video analytics
The “killer” application for edge computing

Other research directions
A vision for general-purpose edge AI systems
Agenda

Mobile computation offloading
The history and motivation behind edge computing

DL-based edge video analytics
The “killer” application for edge computing

Other research directions
A vision for general-purpose edge AI systems
The evolution of mobile devices in the 2000s

Larger screens, more sensors, more sophisticated applications...

Bottlenecks: on-device processing power and battery!
Can we bring network resources to mobile?

Offload the computations of mobile applications to a server (e.g., in the cloud or on-premise) through the network.

“Can you do this for me?”

“Done. Here you go!”

How can we achieve this?
Mobile computation offloading

Client-server paradigm
(Full application or OS)

Part of the application
(Fine-grained code offloading)
Mobile computation offloading: VM-based

Pervasive Computing 2009

Mobile devices employ a VM on a nearby server (called cloudlet) to run applications

What are the limitations of this approach?
Mobile computation offloading: VM-based

High initiation latency (10s seconds), unstable network connection, inflexible (Do we need to offload all? Would the offloading decision the same for all devices)
Mobile computation offloading: code-based

Offloading code from mobile applications to a server over a wireless network transparently

MAUI: Making Smartphones Last Longer with Code Offload

Abstract

The main concern of MAUI is that users may not be aware of how much computation is done on their mobile devices, which can lead to excessive battery consumption. Our goal is to make mobile devices last longer by automatically offloading code to servers. The proposed solution is a code offloading framework that monitors the code execution on mobile devices and offloads it to servers when necessary. The framework is implemented on Android devices and evaluated on various benchmarks. The results show that MAUI successfully offloads code and improves the battery life of mobile devices.

Categories and Subject Descriptors

C.2 [Computer Communication Networks]: Network Systems/protocols

Keywords

mobile devices, battery life, code offloading

CrossCloud: Elastic Execution between Mobile Device and Cloud

Abstract

CrossCloud is a system that allows mobile devices to offload computation to the cloud when necessary. The key idea is to use a lightweight virtual machine to run the mobile application on the cloud, while the mobile device only needs to communicate with the virtual machine. This allows mobile devices to offload computation and conserve battery life.

Test-of-Time Award 2021

COMET: Code Offload by Migrating Execution Temporarily

Abstract

In this paper, we propose COMET, a system that allows mobile devices to offload computation to the cloud. The key idea is to migrate the execution of the mobile application to the cloud temporarily, and then migrate it back to the mobile device when necessary. This allows mobile devices to offload computation and conserve battery life.

Goal: achieve fine-grained code offloading without programmer involvement (optimizing execution time, memory, disk, or energy)
MAUI

Aiming to achieve highly transparent code offloading via method-level migration

Challenges to address: (1) different ISAs on the mobile device (ARM-based) and the server (x86), (2) program state management, (3) offloading decision making
MAUI designs

**Code annotation**  
(Programmer annotates methods/classes that can be executed remotely)

```java
display(...);
read_gps(...);
[remoteable]
video_decoder(...);
```

**Method wrapper**  
(MAUI identifies remoteable methods and generates a wrapper for each of them)

```java
MAUIMessage video_decoder(AppState state, ...){
    ...
    return state;
}
```

**Runtime decision making**  
(Decide to offload or not based on the current network condition)

1. MAUI extracts states at compile time
2. MAUI profiler estimates the cost of executing a method
3. MAUI solver makes offloading decisions with the goals of optimizing performance or energy

Programmer’s involvement is minimized, but still needed.
MAUI workflow

1. Profiling (device, program, network)
2. Running an optimizer to determine the offloading decision
3. State serialization
4. Handover control
5. Remote execution
6. Return control and send back the state

Based on application-level VM (Java VM or .NET)

No support for multi-threaded applications!
Static analysis

Placing migration and re-integration points in the code via static program analysis

```java
class C {
    void a() {
        if() {b();c();}
    }
    void b() {
        // lightweight
    }
    void c() {
        // expensive
    }

    void main() {
        C c; c.a();
    }
}
```

Program

Approximate static control-flow graph

Partitioned graph
CloneCloud
Achieving automatic thread-level mobile code migration **without programmer involvement**

Support for opportunistic concurrency (threads that do not involve offloaded states can continue locally); no support for offloading threads that share the same state
COMET

Achieving submethod-level migration and multi-thread support via distributed shared memory (DSM)

Unmodified & multi-threaded

Mobile application
Memory state
Distribute memory synchronization
PhoneOS

Executes concurrently with non-offloaded threads

Offloaded threads
Memory state
Distribute memory synchronization
RemoteOS

In sync
Via network
Mobile computation offloading did not really fly. What issues were there?
Issues with mobile computation offloading

- **OS support required**
  (Against device makers’ interest)

- **Edge server and network support**
  (A chicken-egg deadlock)

- **Complexity**
  (Need to consider practical issues like mobility)

- **Unclear gains in practical scenarios**
  (PoC is based on controlled environments with selected apps)

Nevertheless, some designs are adopted in current mobile applications implicitly, e.g., Siri.
**Agenda**

- **Mobile computation offloading**
  - The history and motivation behind edge computing

- **DL-based edge video analytics**
  - The “killer” application for edge computing

- **Other research directions**
  - A vision for general-purpose edge AI systems
Video analytics becomes pervasive

Deep neural networks (DNNs) are heavily employed for object detection and recognition.

**Wild-life cameras:** learn about the habit of animals

**Traffic cameras:** monitor traffic conditions

**Drone cameras:** estimate the number of objects

The computation-intensive DNNs are the dominant players in terms of both latency and energy consumption in mobile or IoT applications.
Two general approaches

**On-device:** using a light-weight DNN (e.g., SSD-MobileNet-v2)

**Pros:** low latency, low network traffic, high reliability

**Cons:** suboptimal analytics accuracy

**Platform-based:** using a sophisticated DNN (e.g., FasterRCNN-ResNet101)

**Pros:** high analytics accuracy

**Cons:** high latency, high network traffic
High network variability
Caused by mobility and interference

WiFi network bandwidth variation under static and mobile scenarios

How to handle such high variability?
Research directions

**System adaptation:** conduct tradeoff between latency and accuracy at runtime

Dynamically adjusting the DNN size, input frame size and frame rate to match the real-time bandwidth condition

**Hybrid approach:** combine the processing power of the device and the edge platform

Partition the DNN model across the device and the platform; use a lightweight local path for real-time response
System adaptation
System adaptation

Latency increases almost linearly with the DNN size, but accuracy has a complex context-dependent relationship with the DNN size.

When to adapt the frame rate, and when the frame (DNN) size?
Poor generalization of application-specific solutions

Scenario 1: A surveillance application that detects pedestrians on a busy street

- t=0s, small target in far-field views
- t=1s, small difference
Poor generalization of application-specific solutions

Scenario 2: An application that detects objects on a mobile phone

$t=0s$, nearby and large target

$t=1s$, large difference due to camera movement
Adaptive video stream analytics

Adaptation policies must be

- Precise, automatically generated, for each application

- **Pareto-optimal profile**: maximizing application accuracy while satisfying bandwidth requirements
AWStream

Systematic and quantitative adaptation for video stream analytics

Follow stream processing systems model with new abstraction for specifying tradeoffs

Profile to generate the bandwidth-accuracy statistics and the Pareto-optimal frontier
Adaptation for end-to-end latency guarantee
Automatically adapting the frame size and DNN size to guarantee end-to-end inference latency

A reactive feedback-control based approach

Hybrid approach
Hybrid approach
Combine the local and remote processing power

Local: mobile or IoT devices

Remote: edge server

Solutions:
- Use object tracking to mask network delay
- Partition the DNN into two parts across the local and remote
- Remove the remote processing from the critical path
Hybrid solution #1: object tracking

Use object tracking to mask the delay incurred by edge analytics

Local: mobile or IoT devices

Remote: edge server

Frame sent to the server

Frame at which the result from the server is received

Tracking with optical flow in the middle

Optical flow is not cheap either, so it is only applied on a selected set of frames.

Glimpse (ACM SenSys 2015)
Hybrid solution #2: DNN partitioning
Partition the DNN into two parts, running on the device and the edge server respectively

Local: mobile or IoT devices

Remote: edge server

Intermediate data
First few layers of the DNN
Last few layers of the DNN

Select the partition point based on criteria such as latency/energy minimization

The remote component is in the critical path, so adaptation is still needed!

Neurosurgeon (ACM ASPLOS 2017)
DNN partitioning with early exits

Use early exit points to ensure low latency

Local: mobile or IoT devices

Remote: edge server

Intermediate data

Early exits

Final exit

The system can choose the early exit according to the environmental conditions. The remote component is out of the critical path!
Hybrid solution #3: fusion

Use a lightweight local DNN for fast response and fuse the remote results with the local ones to improve accuracy.

**Local:** mobile or IoT devices

- Lightweight DNN for fast response

**Remote:** edge server

- Full-size DNN for high accuracy
- Accurate but delayed analytics results

Motivation: leveraging temporal correlation

Video stream has **significant temporal correlations** across video frames

- The same content (object/action) may last over multiple frames
- Common frames across **overlapping frames** in window-based analytics like action recognition
Clownfish

A hybrid system for real-time video stream analytics across the device and edge/cloud

Challenges

- **Fuse**: How to combine the device-local analytics results with the remote results from the edge/cloud?
- **Filter**: Which video frames to send to the edge/cloud to achieve bandwidth savings?
Clownfish: system components

Device

- Window manager: generates frame windows
- Local: runs optimized DNN model
- Filter: selects windows to send to edge/cloud
- Fusion: combines the local analytics results with those from the remote

Edge/cloud server

- Remote: runs a complete DNN model
Challenge #1: fusion method

How to combine local results with remote results that might be delayed and intermittent?

A lightweight method that runs on the device

- Exponential Smoothing (ES) approach to fuse past result and current local result
- A weight $\alpha_t$ in ES representing the correlation of the frame windows

Two main procedures

- **Fuse:** used for real-time result fusion
- **Reinforce:** used to update the internal state when remote results become available
Correlation estimation

Define **similarity score** between two consecutive windows

- When score is high, assign relatively larger value for $\alpha_t$, larger weight to the previous state
- Otherwise, rely more on the local result since the previous state may be outdated due to the low similarity

A **learning-based approach** for calculating the similarity score

- Traditional methods like Cosine, Euclidean are not accurate enough

How to set value for weight $\alpha_t$

Already available from the local DNN model, so no extra feature extraction is needed.
Challenge #2: filter method

When to send frame windows to the remote edge/cloud server?

- Send a window at the start of a context: leverage the similarity score to identify context transitions
- Periodically send windows within the same context and restart periodical timer at context transition

Context-aware policy
Clownfish results

Clownfish achieves analytics accuracy comparable to the edge-/cloud-based solution, while maintaining stable throughput to meet real-time requirements.
Agenda

Mobile computation offloading
The history and motivation behind edge computing

DL-based edge video analytics
The “killer” application for edge computing

Other research directions
A vision for general-purpose edge AI systems
Modern distributed computing continuum

Covering a wide range of tradeoffs between latency and service quality
Edge storage systems
Collaborative edge applications

Edge infrastructure
- Distributed: cloud-edge continuum
- Heterogeneous: servers, workstations, Jetson boards
- Dynamic: user mobility, resource reclamation

Edge applications: many are collaborative
- AR/VR/MR: user profile, game state
- Autonomous driving: maps, LiDAR data, models
- IoT sensing/analytics: environment, tracking state
- Edge ML/DL: shared models/parameters, training data
Current cloud computing paradigm

**Principle**

Decouple compute and storage for higher scalability and availability, and lower cost

**Cloud solutions**

HDFS, Amazon S3, Redis, Cassandra...

**Serverless computing:** function as a service

compute requests → stateless functions → application state/data → storage service

- Amazon S3
- Redis
How to apply it to the edge?

Question: Can we just use cloud storage solutions at the edge?

Short answer is **NO**, because of the new challenges (distributed, heterogeneous, dynamic) imposed by the edge

- **High latency for strong consistency** (multi-RTT)
- **High cross-site traffic volume**
Edge storage

Abstractions/APIs
- KV-pairs, graph-based, time-series

Locality
- Replication, spatio-temporal encoding

Heterogeneity
- Partial replica, TTL-based data eviction

Mobility
- Session migration, replica placement

Failover
- Zones, erasure encoding, CRDT

Scalability
- Spatio-temporal hashing

Semantics
- Context-/Location-based, consistency

Monitoring
- Resource usage, dynamics, mobility
Edge storage: state of the art

<table>
<thead>
<tr>
<th>Abstraction/API</th>
<th>Locality</th>
<th>Heterogeneity</th>
<th>Mobility</th>
<th>Failover</th>
<th>Scalability</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>PathStore</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>session/eventual</td>
</tr>
<tr>
<td>FogStore</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>context-aware</td>
</tr>
<tr>
<td>DataFog</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>eventual</td>
</tr>
<tr>
<td>RedWedding</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>conflict-free</td>
</tr>
<tr>
<td>DPaxos</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>quorum-based</td>
</tr>
<tr>
<td>EdgeCons</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>quorum-based</td>
</tr>
<tr>
<td>TSDBs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>range, aggregate</td>
</tr>
<tr>
<td>Cachier</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td>Vision-specific</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>N/A</td>
</tr>
</tbody>
</table>

- ✔ Full support
- ○ Partial support
- ✘ No support
- ! Unknown

Various abstractions and semantics

Little to no support
A multi-consistency hierarchical distributed storage service for edge computing

1. **Multi-consistency declarative API**
   - Tradeoffs between latency and consistency [Terry et al. SOSP’13]
   - Timestamp-based conflict resolution
   - Reduce (de)serialization cost

2. **Model-based resource management**
   - Graph-based models for heterogeneous resources
   - Adaptive optimization mechanisms

3. **Real-time monitoring**
   - Infrastructure-centric latency/resources monitoring
   - Mobility monitoring/prediction

Edge resource management
Edge resource management

Edge resource management: workload migration

Contributions: bounded-performance dynamic resource allocation algorithm under arbitrary user mobility

Edge-centric programming
Assume we want to develop an IoT application involving three entities:

- **IoT sensor**: low-level C programming on microcontrollers
- **Cloud**: advanced programming paradigms with serverless computing/microservice
- **Mobile**: medium-level programming with languages like Java

Current mentality: the application has to be split onto different platforms and programmed separately → high complexity and high maintenance cost
An edge-centric programming model

Challenges: (1) heterogeneity in native code generation and migration, (2) distributed state management, (3) resource scheduling
Summary

Proliferation of modern mobile ubiquitous applications
- Computation-intensive, require low latency and high reliability
- Modern distributed computing continuum

The evolution of edge computing
- Mobile computation offloading and cloudlets
- Mobile code offloading solutions

The “killer” application for edge computing: DL-based edge video analytics
- System adaptation
- Hybrid approach: mobile tracking, DNN partitioning and early exit, and fusion

Other research directions
- Edge storage, edge resource management, and edge-centric programming