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

Arun Ravindran and Anjus George
UNC Charlotte

Distributed Vision at the Edge - Smart City

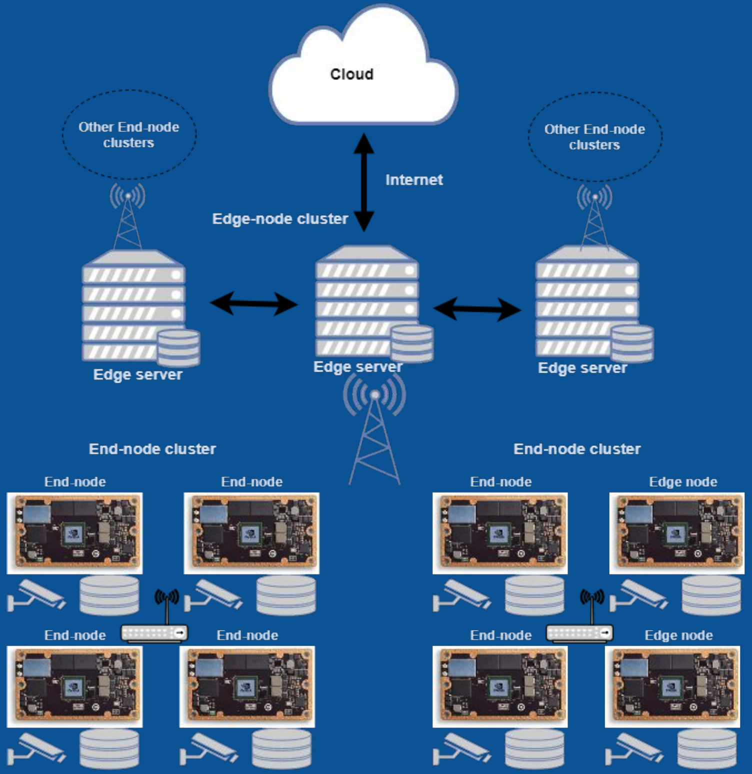
Source: YouTube

Warn pedestrian about potential accidents

Automatically detect and alert drunk driving

Effective bias free law enforcement

System Architecture



Cameras

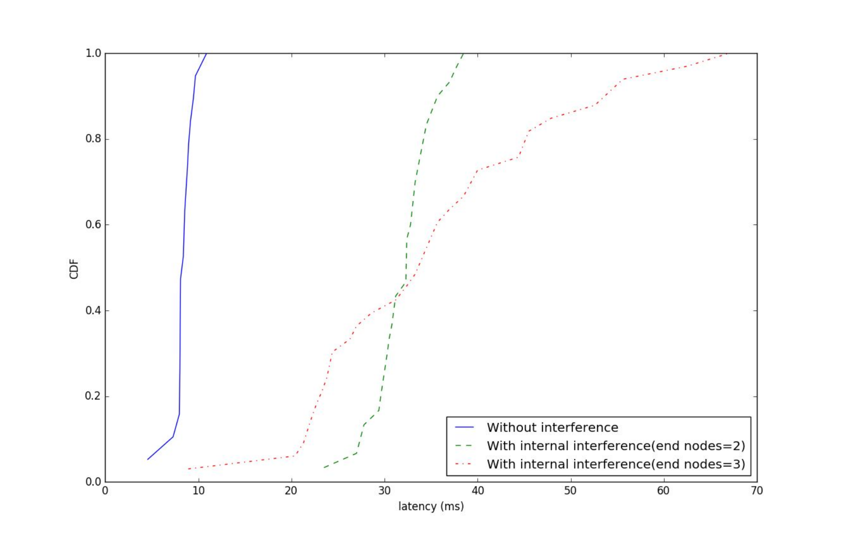
End nodes

Edge Servers

Cloud

Vision Edge Datastore

- Applications at Edge analyzes data collected by End nodes to detect events
  - Need data store that persists data gathered from multiple end nodes
  - Able to specify latency required
- Challenge - how to maintain low latency at edge ?
  - Latency sources - wireless channel, bufferbloat, read/write latency



Latency CDF witn node scaling

Cloud vs Edge

- Data Center vs. “Field”
  - Security, Fault tolerance
- Wired vs. Wireless
  - Bandwidth, latency
- Homogeneous vs. Heterogeneous
  - ARM/x86 SoCs, Multiple storage and networking technologies
- Distributed data storage vs. Distributed data at source
  - Big, fast, distributed data
  - latency critical/sensitive applications

Prior Work at Edge Storage

- VisFlow Project (Microsoft)
- PathStore Project (Toronto)
- Cachier Project (CMU)

Our Design philosophy at Edge

- Application specific systems
  - Tension between specificity and generality
- Autonomous operation
  - Techniques from Control Theory and AI (ML, Deep Learning, Reinforcement Learning)

Key idea - Exploit application characteristics

- Two type of data - image feature vectors (1-10 kB) and image keyframes (100 - 500 MB)
- Feature vectors - latency critical
  - Tracking, behavioral analysis
- Keyframes - latency sensitive
  - Archival
- Feature vector latency by sacrificing keyframe accuracy
  - Need to do this dynamically since channel interference and scene content is dynamic

Key idea - Latency control knobs

- Control knob 1: Keyframe TX
  - Controls the rate at which keyframes are transmitted
  - Low egress rates could result in bufferbloat
- Control knob 2: Keyframe Sim
  - Drops similar keyframes to maintain buffer length
  - Accuracy vs. Latency trade off
  - Needs a similarity metric

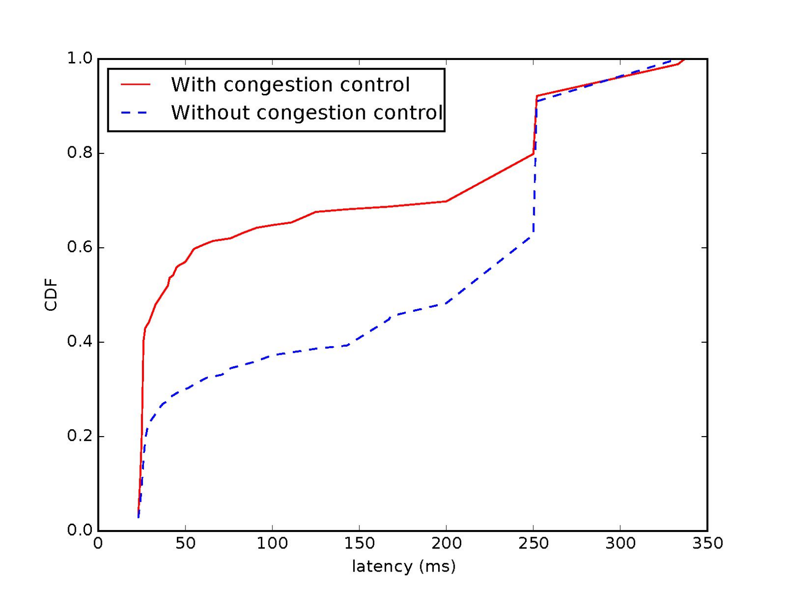
Vision Edge Data Store - Design

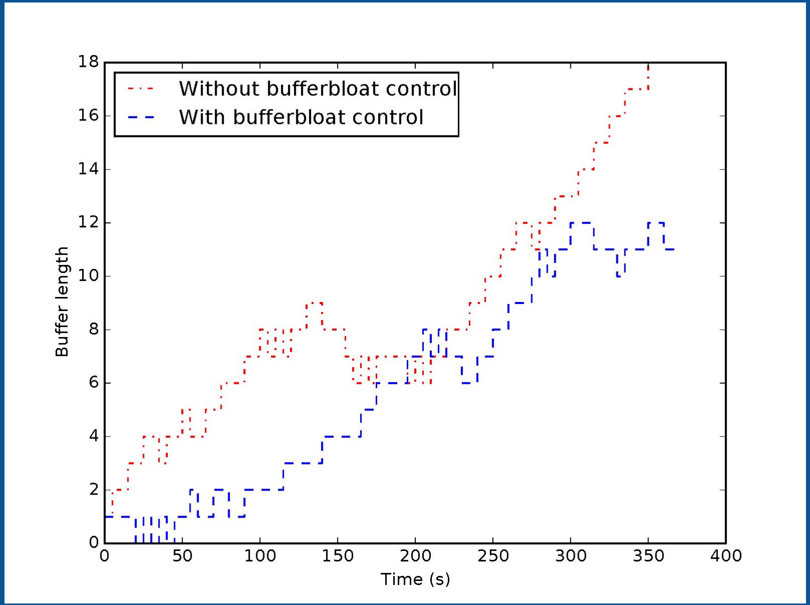
- End node processing generates key frames feature and feature vectors
- Inserted with timestamp and node ID into transmit buffer
- Data transmitted to Edge server
- Aggregate and persist data at Edge server
  - Low latency store (RocksDB, RAMCloud)
- End node controls keyframe Tx rate and buffer length
  - Scalable since controllers are independent

Prototype Evaluation Platform

- Emulation platform
  - LXC containers for nodes
  - NS3 network simulator for WiFi channel
  - Client/Servers implemented in Golang
  - Image similarity (SSIM) with Python sckit-image
  - qperf for latency measurements
- Controller
  - Bang-bang (on/off) control
- Data
  - 500kB keyframes, 4 kB feature vectors
  - External interference simulated via Poisson process (5s TX, λ = 30s)

Results

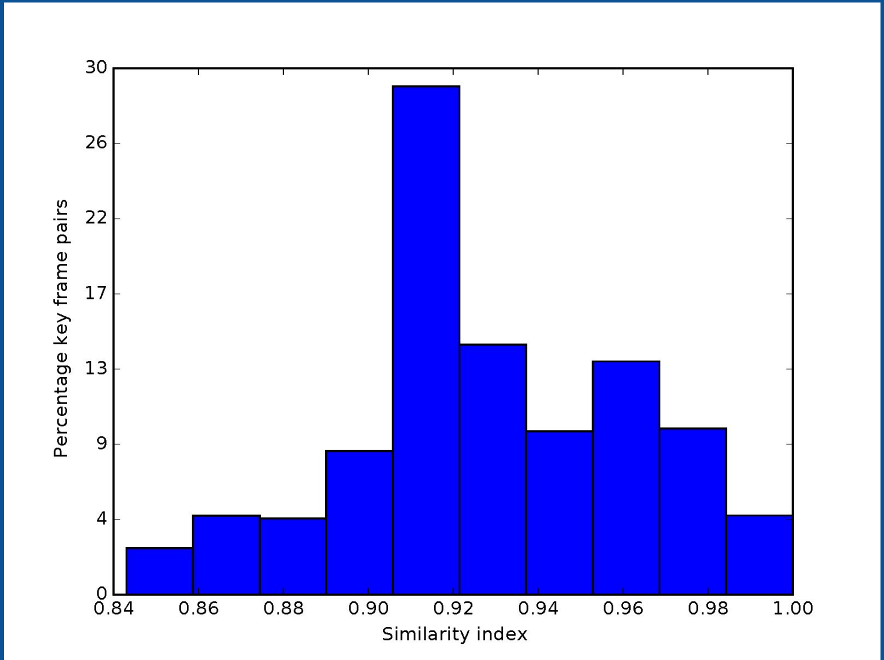




Latency CDF - Keyframe TX control

Latency CDF - Keyframe Sim control

Keyframe similarity (SSIM) - Pedestrian crash video



Accuracy vs. Latency tradeoffs

On going work

- Experimental characterization of interference, keyframe similarity, application requirements
- Internal interference - scheduling problem?
  - Distributed - client-server vs. peer-to-peer
  - Dependence on scene dynamics
- Control / Learning algorithms

Request Feedback

- On use of WiFi at Edge for latency critical apps?
- On differences between cloud and edge storage?
- What would you like to see experimentally validated?
- How should latency/accuracy requirements be communicated from Edge app. to camera end nodes?
- Are there other edge applications that are similar?
- What edge specific security issues should we consider?
- Any experience with simulating NS3 802.11ac with containers?