HIVEMIND: A HARDWARE-SOFTWARE SYSTEM STACK FOR SERVERLESS EDGE SWARMS

Liam Patterson, David Pigorovsky, Brian Dempsey, Nikita Lazarev, Aditya Shah, Clara Steinhoff, Ariana Bruno, Justin Hu, and Christina Delimitrou

Cornell University

sail.ece.cornell.edu
Executive Summary

- **Edge swarms increasing in size & complexity:**
  - Enable new IoT applications
  - Require rethinking the cloud-edge system stack

- **Challenges:**
  - Programming interface → abstract away system/app complexity
  - Execution environment → fine-grained, event-driven tasks
  - Hardware design → network communication, computation, etc.

- **HiveMind: end-to-end hardware-software stack for cloud-edge systems**
  - Declarative programming interface, automated task/data placement
  - Serverless execution environment, reconfigurable hardware acceleration
  - Significant performance, efficiency, programmability gains vs. centralized and decentralized platforms
All Computing Involves the Cloud

From few large applications...
All Computing Involves the Cloud

From few large applications to every service from every domain of human activity.
Many Apps Running on Low-Power Edge Swarms

- Wearables
- Navigation
- Robotics
- Drones
- ML
- Analytics
- Self-driving cars
- IoT
- Space
Many Apps Running on Low-Power Edge Swarms

Wearables

Navigation

Robotics

Drones

Disaster recovery
Digital agriculture
ML analytics
Navigation, etc.

Interactive, latency-critical tasks
Must meet user-provided QoS
Minimize battery consumption
Address failures

ML

Analytics

Self-driving cars

VR

IoT

Space
Programming framework

Abstract the complexity of managing a cloud-edge system from the end user.
Cloud-Edge System Stack Requirements

Programming framework

Task & data placement

Abstract the complexity of managing a cloud-edge system from the end user

Automatically determine where computation & data should be placed to meet QoS, handle failures
Cloud-Edge System Stack Requirements

Programming framework

Abstract the complexity of managing a cloud-edge system from the end user

Task & data placement

Automatically determine where computation & data should be placed to meet QoS, handle failures

Execution environment

Match the fine-grained, event-driven and intermittent nature of IoT tasks
Cloud-Edge System Stack Requirements

- **Programming framework**
  - Abstract the complexity of managing a cloud-edge system from the end user

- **Task & data placement**
  - Automatically determine where computation & data should be placed to meet QoS, handle failures
  - Match the fine-grained, event-driven and intermittent nature of IoT tasks

- **Execution environment**
  - Identify resource bottlenecks, enable reconfiguration to match IoT apps’ frequent updates

- **Hardware design**
HiveMind Design

- **Hardware-software system stack for cloud-edge systems**
  - Focus on low-power edge devices
  - Focus on multi-phase computation that requires data transfer across phases

- **Methodology:**
  - Focus on programmable drones: Parrot A.R. Drone 2.0
    - ARM core on-board, 4GB of memory + 24GB of SSD
    - Generalizes to other swarms
  - 20-server backend cloud with 2-socket Intel servers

- **Applications:**
  - Single-tier tasks (face recognition, weather analytics, SLAM, obstacle avoidance, etc.)
  - Multi-tier scenarios (treasure hunt, people search)
HiveMind Programming Model

Declarative Programming Interface (HiveMind DSL)
User expresses \{task graph, i/o, task logic\} in Python
End-to-end source, and inter-task APIs synthetized automatically

Task graph

```
TaskGraph(list=['createRoute', 'collectImage', 'obstacleAvoid', 'faceRecognition', 'deduplication'],
constraint=[execTime='10s'])
```
HiveMind Programming Model

Declarative Programming Interface  (HiveMind DSL)
User expresses {task graph, i/o, task logic} in Python
End-to-end source, and inter-task APIs synthetized automatically

Task graph

Task description

Create Route
Collect Image
Obstacle Avoidance
Face Recognition

Task(collectImage,None,sensorData,
filepath/to/task/code,
speed='4',resolution='1624p',
colorFormat='color',
parentTask=['createRoute'],childTask=[
'obstacleAvoidance','faceRecognition'])
HiveMind Programming Model

Declarative Programming Interface  (HiveMind DSL)
User expresses \{task graph, i/o, task logic\} in Python
End-to-end source, and inter-task APIs synthetized automatically

Task graph

Task description

```python
Task(faceRecognition, sensorData, recognitionStats,
    'filepath/to/task/code',
    trainingData='zoo',
    algorithm='tensorflow_zoo',
    parentTask=['collectImage'],
    childTask=['deduplication'])
```
HiveMind Programming Model

Declarative Programming Interface (HiveMind DSL)

User expresses \{task graph, i/o, task logic\} in Python
End-to-end source, and inter-task APIs synthetized automatically

Task graph

Task description

```
Task('deduplication, recognitionStats, dedupList,
    'filepath/to/task/code',
    sync='all',
    parentTask=['faceRecognition'],
    childTask=[])```

Create Route

Collect Image

Obstacle Avoidance

Face Recognition

Face Deduplication
HiveMind Programming Model

Declarative Programming Interface (HiveMind DSL)

User expresses {task graph, i/o, task logic} in Python
End-to-end source, and inter-task APIs synthetized automatically

Task graph
Task description
Properties & constraints

Create Route
Collect Image
Obstacle Avoidance
Face Recognition
Face Deduplication

- Parallel(obstacleAvoidance, faceRecognition)
- Serial(faceRecognition, deduplication)
- Learn(faceRecognition, 'Global')
- Place(obstacleAvoidance, 'Edge:all')
- Persist(faceRecognition)
- Persist(deduplication)
HiveMind Programming Model

Declarative Programming Interface (HiveMind DSL)
User expresses \{task graph, i/o, task logic\} in Python
End-to-end source, and inter-task APIs synthetized automatically

Output
end-to-end application code base, and inter-task APIs for different placement options (cloud, edge, hybrid)
HiveMind Scheduler

Automated Task & Data Placement

- Cloud execution is usually faster, not always.
- Performance variability is higher at the edge.
- Cloud offloading quickly saturates network link.
HiveMind Scheduler

Automated Task & Data Placement
Explores division of tasks across cloud and edge resources to meet end-to-end QoS (performance and/or power) constraints
HiveMind Scheduler

Automated Task & Data Placement
Explores division of tasks across cloud and edge resources to meet end-to-end QoS (performance and/or power) constraints

Create Route

Latency & Power Measurement

Collect Image

Latency & Power Measurement

Obstacle Avoidance

Obstacle Avoidance

Face Recognition

Face Recognition

Face Deduplication

Face Deduplication

Latency & Power Measurement

Latency & Power Measurement

Latency & Power Measurement
HiveMind FaaS Environment

Serverless Execution Environment
Leverages FaaS to offload computation to the cloud

- Much lower latency compared to cost-equivalent reserved resources
- Adjusts much better to load fluctuations
- Handles failures more smoothly
HiveMind FaaS Environment

Serverless Execution Environment
Leverages FaaS to offload computation to the cloud

Higher latency variability

High instantiation & control plane overheads
High data transfer overheads
HiveMind FaaS Environment

Serverless Execution Environment
Leverages FaaS to offload computation to the cloud

- Centralized cloud controller implemented in OpenWhisk:
  - Global visibility into cloud and edge resources
  - Pre-warms containers and caches images with high reuse probability
  - Places dependent functions physically close (same container or same node)
  - Motivates need for hardware support for remote memory access

23
HiveMind Hardware Design

Hardware design
Accelerates communication between cloud-edge and within cloud tasks

often becomes the performance bottleneck
Worse for ML-heavy tasks
Worse for unreliable network connectivity
HiveMind Hardware Design

Hardware design
Accelerates communication between cloud-edge and within cloud tasks

- Two reconfigurable acceleration fabrics:
  - Cloud-edge communication $\rightarrow$ RPC acceleration
  - Cloud-cloud function communication $\rightarrow$ RDMA acceleration
  - Implemented in a tightly-coupled cache coherent FPGA (NUMA interconnect, UPI bus)
  - Spatially partitioned, supports multi-tenancy and resource isolation
**HiveMind System Stack**

- **Implementation:**
  - ~28,000 LoC (C++, Python, node.js, Verilog, VivadoHLS)
  - Centralized controller
    - Hot stand-by copies
  - End-to-end monitoring system (minimal perf overhead)

- **Other features:**
  - Fault tolerance → load rebalancing
  - Straggler detection
  - Online learning → per-device, swarm-wide

- **Comparisons:**
  - Fully centralized system
  - Fully decentralized system
  - With and without serverless
Evaluation: Performance

- **Task/Job latency:**
  - Lower than both centralized and distributed
  - More predictable (less variability)
  - Mostly benefits multi-tier compute-/data-intensive jobs
Evaluation: Power Consumption & Net Bandwidth

- **Power consumption:**
  - 73% lower power consumption than distributed
  - 18% lower power consumption than centralized

- **Network bandwidth:**
  - 78% lower bandwidth utilization than centr.
  - ~3x higher bandwidth utilization than distributed
- Task/Job latency:
  - Modular design → performance & efficiency can benefit from subset of techniques
  - But all techniques are needed to achieve best performance and efficiency
Other Experiments (in the paper)

- Latency breakdown
- Fault tolerance
- Scalability with swarm size, resource requirements
- Portability to other swarms (robotic cars)
- Online learning
- Etc.
Conclusions

- **Edge swarms increasing in size & complexity:**
  - Enable new IoT applications
  - Require rethinking the cloud-edge system stack

- **Challenges:**
  - Programming interface → abstract away system/app complexity
  - Execution environment → fine-grained, event-driven tasks
  - Hardware acceleration → network communication, computation, etc.

- **HiveMind: end-to-end hardware-software stack for cloud-edge systems**
  - Enables programmable cloud-edge platforms
  - Automates task and data placement
  - Leverages serverless compute and reconfigurable hardware acceleration
  - Offers significant performance and efficiency gains vs. centralized and decentralized platforms
Questions?

- Edge swarms increasing in size and complexity:
  - Enable new IoT applications
  - Require rethink of cloud-edge system stack

Challenges:
- Programming interface: abstract away system/app complexity
- Execution environment: fine-grained, event-driven tasks
- Hardware acceleration: network communication, computation, etc.

HiveMind: end-to-end hardware-software stack for cloud-edge systems
- Enables programmable cloud-edge platforms
- Automates task and data placement
- Leverages serverless compute and reconfigurable hardware acceleration
- Offers significant performance and efficiency gains vs. centralized and decentralized platforms