Challenges in Edge Computing I: Programmability, Naming & Resource Optimization

Chapter 5

Topic: Programmability | Naming & Data | Resources | Metrics

Outline

- ❑ Programmability
  - Firework, OpenStack++
  - FocusStack, SpanEdge
  - Partitioning: PDG, static/dynamic
- ❑ Naming & Data
  - DNS limits at the edge
  - NDN & MobilityFirst
  - edgeOS naming + data abstraction
- ❑ Resource Allocation
  - Scheduling strategies
  - Data offloading
  - Load balancing
- ❑ Optimization Metrics
  - Latency & bandwidth
  - Energy consumption
  - Cost trade-offs
- ❑ Summary & Practice

Programmability: Challenge

Why edge is hard to program

Cloud model

Infrastructure is transparent

- Write once, deploy to the cloud
- One homogeneous target platform
- Provider maintains the servers
- User need not know the internals

Edge model

Programmer bears the complexity

- Tasks migrate from cloud to edge nodes
- Heterogeneous platforms & runtimes
- MapReduce & Spark don’t fit
- Needs new programming models

Programmability: Programming Models

Four frameworks that tame the edge

➢ Heterogeneous edges need purpose-built programming models. Four influential frameworks — each explored next.

01 **Firework**

Zhang et al.

A virtual shared data view over distributed sources; computation flows cut what's transmitted.

02 **OpenStack++**

Ha & Satyanarayanan

A programming model for the cloudlet architecture, tailored to mobile environments.

03 **FocusStack**

Amento et al.

Finds edge devices with enough resources, then deploys & tracks apps across mobile IoT devices.

04 **SpanEdge**

Sajjad et al.

Unifies cloud & near-edge nodes; developers mark which parts run near the data source.

Quan Zhang et al. "Firework: Big data sharing and processing in collaborative edge environment". In: 2016 Fourth IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb). IEEE. 2016, pp. 20–25.

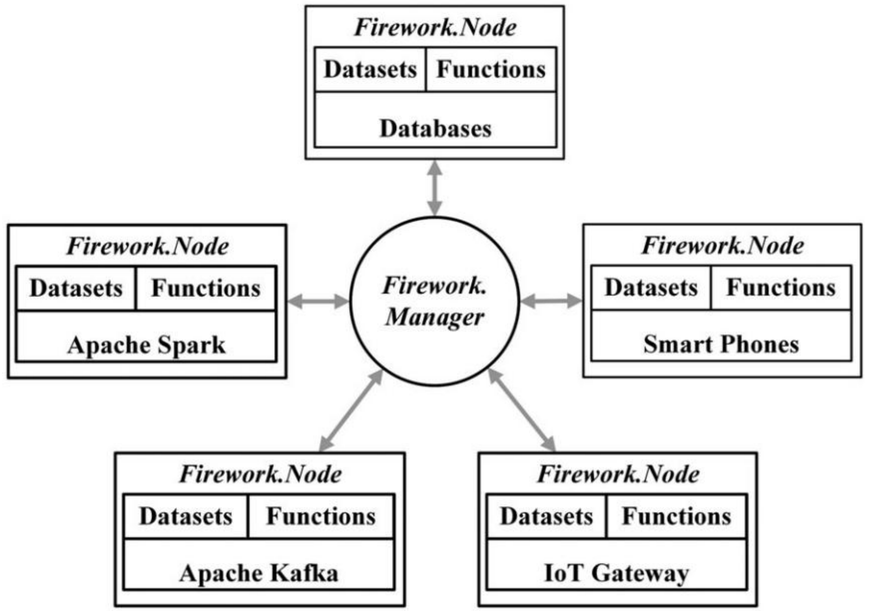
Kiryong Ha and Mahadev Satyanarayanan. OpenStack++ for cloudlet deployment. Technical report CMU-CS-15-123. School of Computer Science, Carnegie Mellon University, 2015.

Brian Amento et al. "FocusStack: Orchestrating edge clouds using location-based focus of attention". In: 2016 IEEE/ACM Symposium on Edge Computing (SEC). IEEE. 2016, pp. 179–191.

Hooman Peiro Sajjad et al. "SpanEdge: Towards unifying stream processing over central and near-the-edge data centers". In: 2016 IEEE/ACM Symposium on Edge Computing (SEC). IEEE. 2016, pp. 168–178.

Programmability: Firework

A programming model for big-data sharing & processing in a collaborative edge environment.



Edge computing paradigm: the Firework model (managers + nodes).

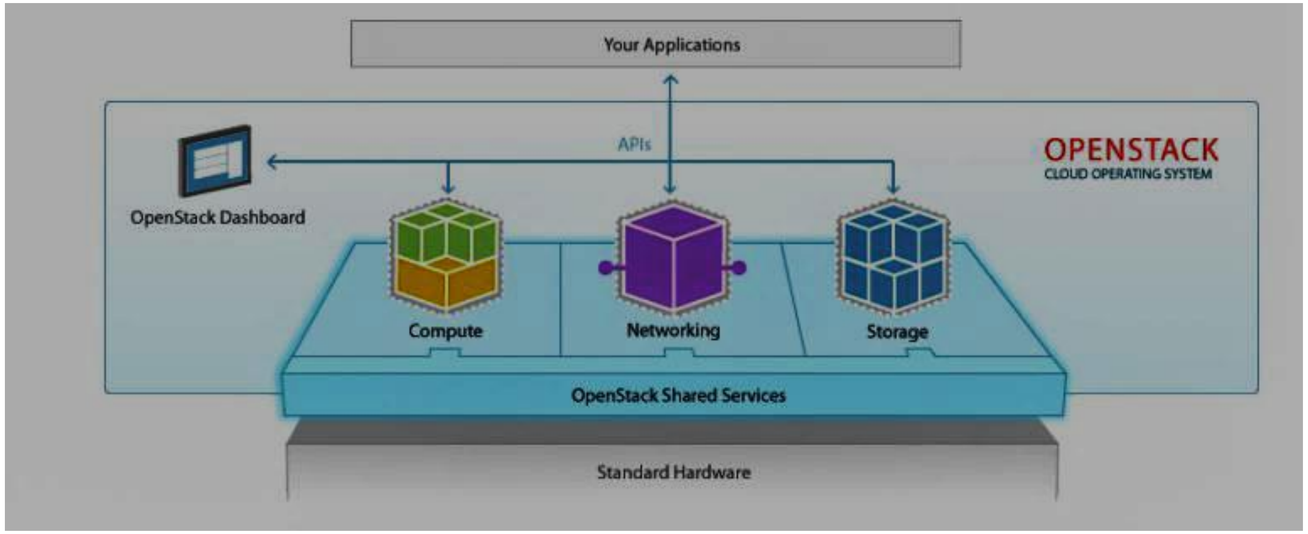
➢ Key objectives

- **Virtual shared data view** Fuse distributed data sources behind predefined interfaces.
- **Datasets + functions** Each node exposes accessible datasets and functions on them.
- **Privacy-preserving** Functions share only knowledge — output ≥1000× smaller than raw.
- **Computation flows** Process data along its path to cut what's transmitted.

Quan Zhang et al. "Firework: Big data sharing and processing in collaborative edge environment". In: 2016 Fourth IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb). IEEE. 2016, pp. 20–25.

Programmability: Openstack++

A programming model for the cloudlet architecture, tailored to mobile environments.



OpenStack Software Overview Diagram (Source)

➢ Key objectives

Cloudlet deployment Extend OpenStack APIs to provision nearby cloudlets.

VM-based provisioning Package apps as VM overlays pushed to the cloudlet.

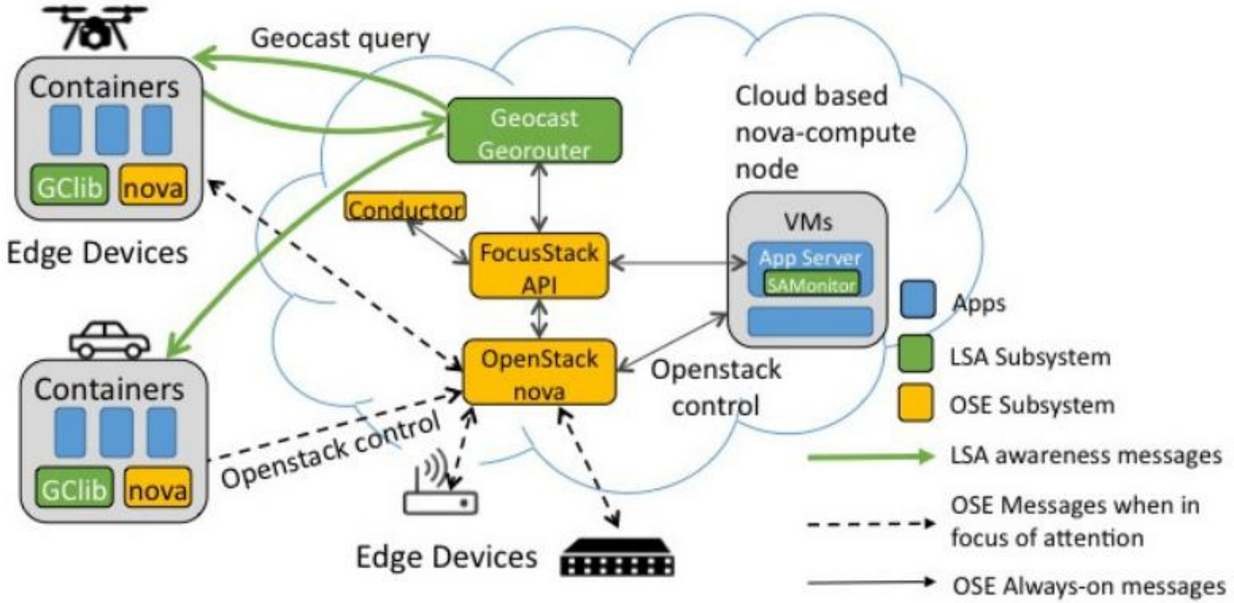
Low-latency offload Mobile apps offload compute to a one-hop cloudlet.

VM handoff Migrate the VM as the mobile user moves.

Kiryong Ha and Mahadev Satyanarayanan. OpenStack++ for cloudlet deployment. Technical report CMU-CS-15-123. School of Computer Science, Carnegie Mellon University, 2015.

Programmability: FocusStack

Deploy diverse, complex apps across mobile IoT edge devices — as an IaaS cloud.



FocusStack Architecture (Source)

Location-based situation awareness (LSA)

Query only healthy devices in the target area ⟶ deploy containers to them ⟶ track just those in focus

➢ Key objectives

Location-based awareness LSA over a geographic addressing network.

Focus of attention Only track & message healthy devices in the target area.

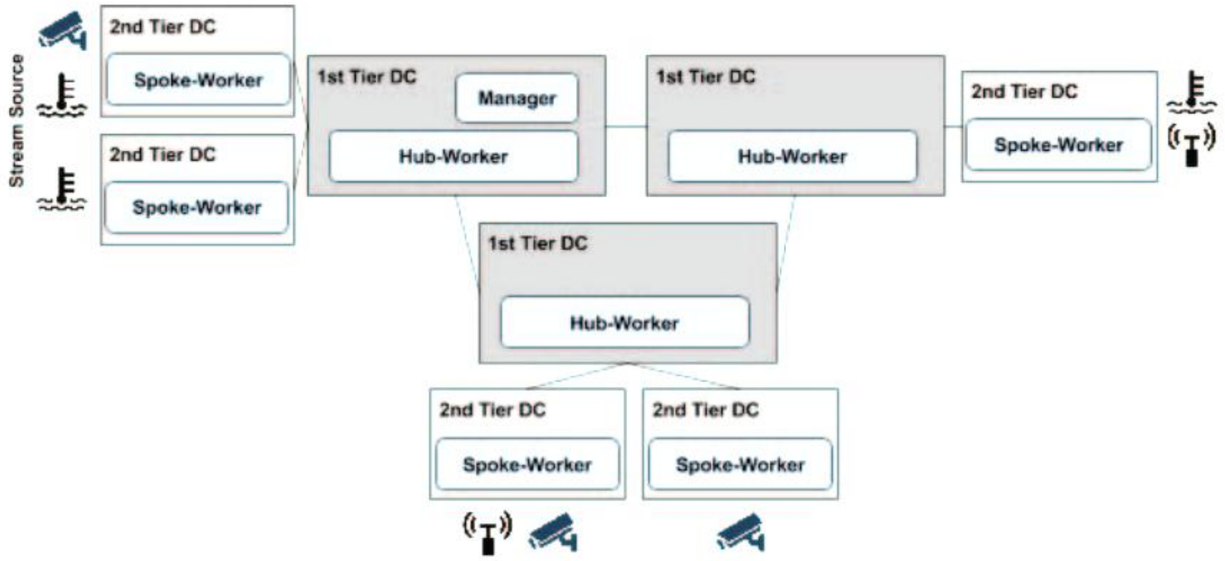
Container deployment Place containers onto in-focus devices with enough resources.

Efficient control plane Untracked devices don't drain resources or messaging.

Brian Amento et al. "FocusStack: Orchestrating edge clouds using location-based focus of attention". In: 2016 IEEE/ACM Symposium on Edge Computing (SEC). IEEE. 2016, pp. 179–191.

Programmability: SpanEdge

Unify stream processing across central and near-the-edge data centers.



SpanEdge Architecture (Source)

Local tasks run near the source

Master–worker: a Manager schedules tasks onto hub- and spoke-workers.

➢ Key objectives

Two-tier workers Hub-workers in the cloud (T1); spoke-workers at the edge (T2).

Local vs. global tasks Local tasks run near the source; global tasks aggregate results.

Minimize WAN latency Scheduler places tasks to cut wide-area network delay.

Source-agnostic dev Developers mark what runs near data, not where data lives.

Hooman Peiro Sajjad et al. "SpanEdge: Towards unifying stream processing over central and near-the-edge data centers". In: 2016 IEEE/ACM Symposium on Edge Computing (SEC). IEEE. 2016, pp. 168–178.

Programmability: Program Partitioning

Splitting apps across edge nodes

- A standalone app must be decomposed into components — preserving semantics — and placed across heterogeneous cloud-edge and edge-edge nodes, weighing resources, energy and latency.

Static partitioning

Decided at compile time

Fixed at compilation — as in MPI programming and GPGPU-based heterogeneous multicore computing.

Dynamic partitioning

Decided at run time

Adapts during execution to changing resources, network conditions and node availability.

Program Dependence Graph (PDG).

Keeps the app's component nodes but adds edge-specific weights — heterogeneous resource availability, location distance, communication cost — and new dependencies: resource-availability, location-mobility and response-time. Drives partitioning across static, dynamic, cloud-edge and edge-edge phases.

Naming & Data: Naming Convention

Naming billions of moving devices

Why DNS & IP fall short.

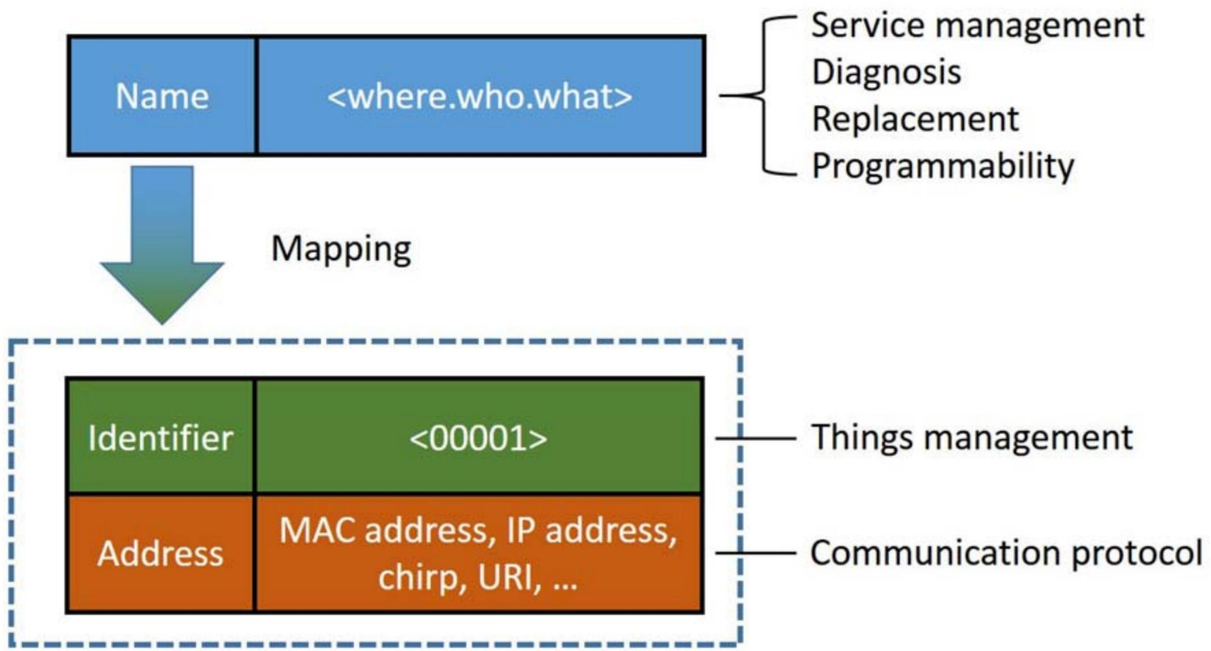
Traditional DNS / URIs lack the flexibility for dynamic edge networks; IP-based schemes are too heavy for resource-constrained, highly mobile devices.

NDN

Named Data Networking. Hierarchical, content-centric names; scalable & human-friendly but bridging to Bluetooth/ZigBee needs proxies & raises isolation concerns.

MobilityFirst

Separates names from addresses. Great for mobility, but its global unique identifier (GUID) is impractical for fixed homes and unfriendly for service management.



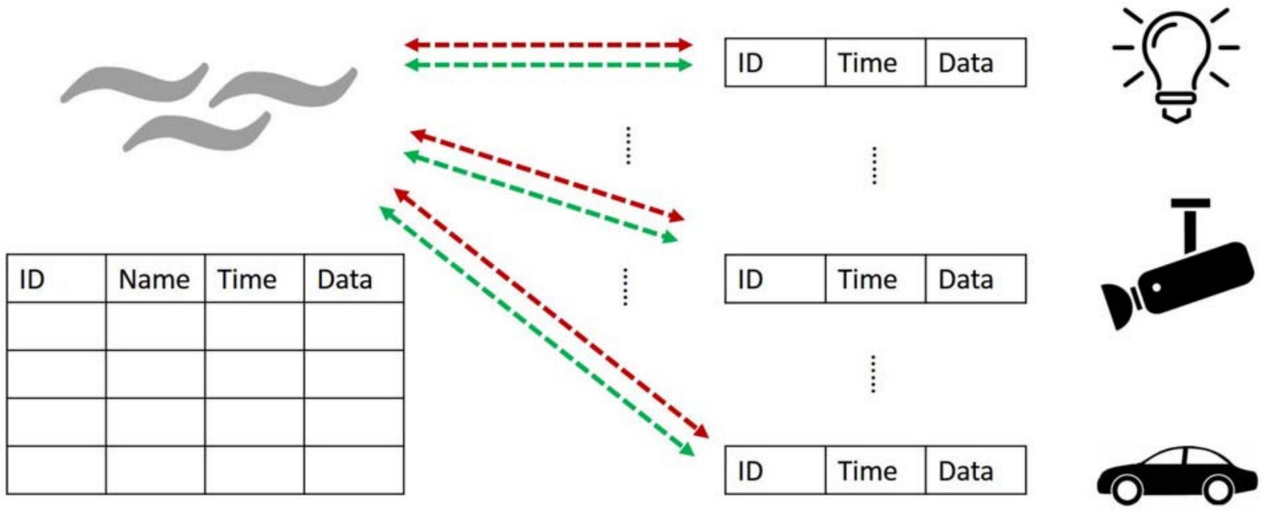
edgeOS naming: a human-friendly name (where.who.what) maps to identifiers & addresses (Shi et al.).

edgeOS naming. Names carry location, role & data description — e.g. “kitchen.oven2.temperature3” — simplifying diagnosis, replacement and privacy while mapping to IP/MAC underneath.

Weisong Shi et al. “Edge computing: Vision and challenges”. In: *IEEE Internet of Things Journal* **3**. 5 (2016), pp. 637–646.

Naming & Data: Data Abstraction

Three challenges of abstracting edge data



Data abstraction in edge computing: heterogeneous devices → a unified data table

Gateways preprocess data (denoising, event detection, privacy) into a standardized table — e.g. 0000, 12:34:56 PM, 01 Jan 2016, kitchen.oven2.temperature3, 78 — before serving upper layers.

Diversity of formats

Devices emit different formats; a unified table hides raw specifics for privacy — but that also hides detail apps may need.

Degree of abstraction

Filter too much and services starve; keep too much and storage suffers — and edge data is often unreliable.

Applicability

Apps need read/write access, but heterogeneous device representations & operations block a truly general abstraction.

Weisong Shi et al. “Edge computing: Vision and challenges”. In: *IEEE Internet of Things Journal* **3**. 5 (2016), pp. 637–646.

Resource Allocation: Scheduling

Mapping tasks to scarce resources

Graph-theory formulation.

Represent each app as a graph (nodes = components, edges = communication) and resources as a graph (nodes = servers). Scheduling becomes a mapping between the two, adapting as resources, network and user location change.

- *Must also handle heterogeneity, application diversity, and the overhead of user mobility while maximizing utilization and provider benefit.*

- **Three delay-based strategies**

Shortest transmission time first

Pick the node that moves the task's data fastest.

Shortest queue length first

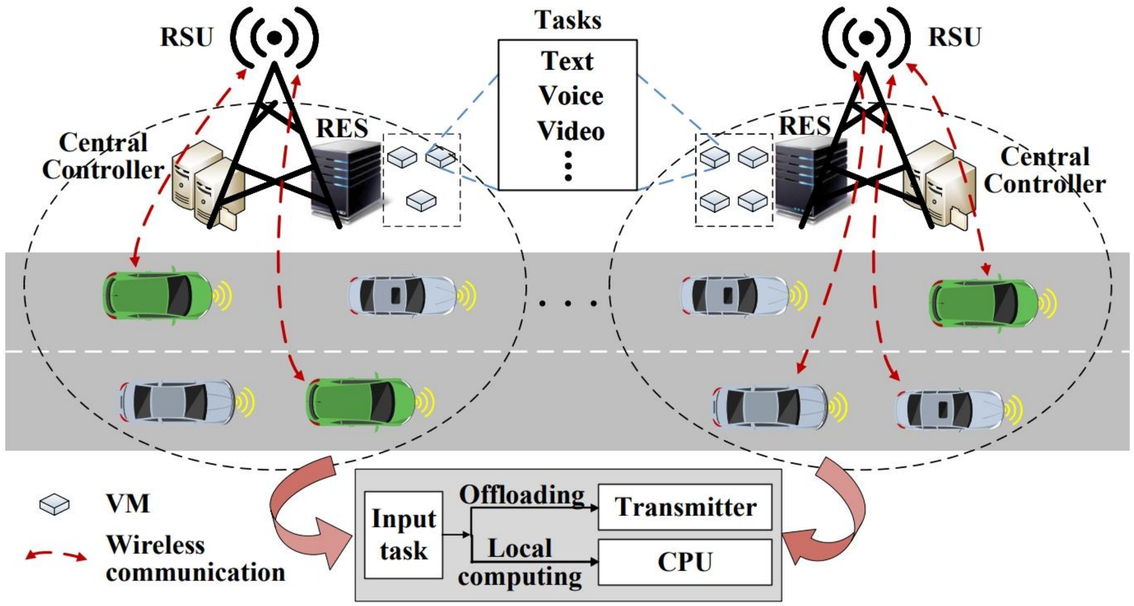
Send to the least-backed-up node.

Shortest scheduling delay first

Minimize total time before execution starts.

All three exploit the edge's core advantage: lower data-transmission latency.

Resource Allocation: Offloading & Load Balancing



Offloading framework for vehicle-edge collaboration

Load balancing

Distribute workload across far/mid/near-edge layers, keep links stable, and use AI + SDN to orchestrate routing across the whole edge network.

➢ Two offloading modes

Full offloading

Move the whole task to the edge/server. Optimize for delay (if energy allows) or minimize energy under a delay constraint, using queue, resource-utilization & routing info.

Partial offloading

Split the task into blocks; offload only selected ones. Choice depends on data volume, device utilization, channel conditions & energy — jointly optimizing comms + compute.

Quyuan Luo et al. “Minimizing the delay and cost of computation offloading for vehicular edge computing”. In: IEEE Transactions on Services Computing 15. 5 (2021), pp. 2897–2909.

Optimization Metrics

Different layers have different capabilities, so where a load runs is a design choice — evenly spread, piled on one layer, or pushed entirely to edge or cloud. Four metrics decide:

Latency

Run at the nearest capable layer — but the closest layer isn't always optimal if it's busy.

Bandwidth

High bandwidth cuts transmission delay; local preprocessing shrinks the upload volume.

Energy

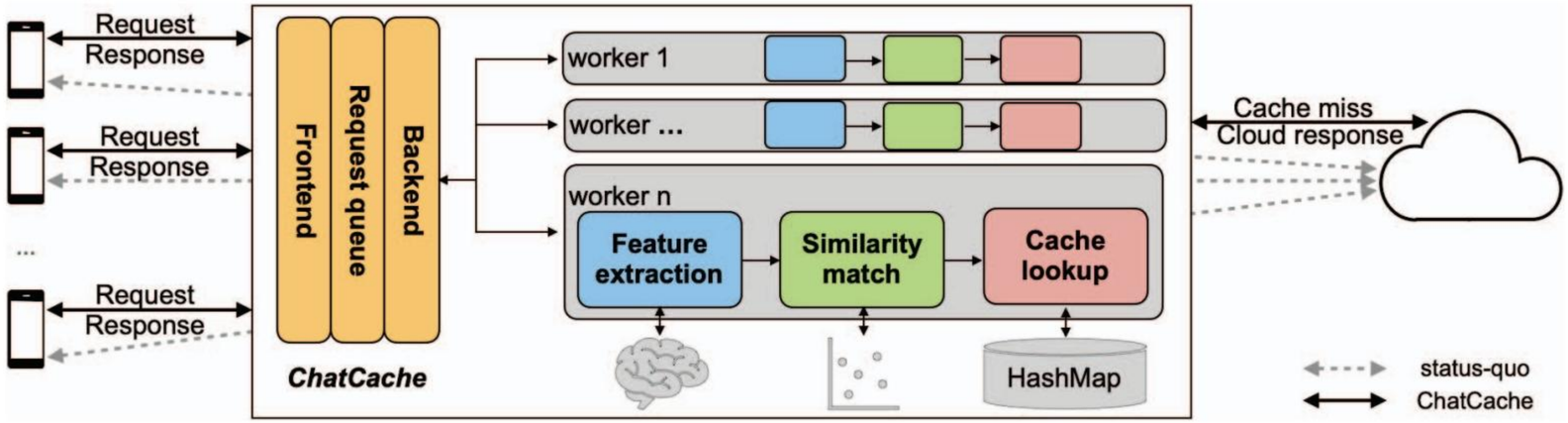
Offload when transmission energy < local compute energy; the battery is the tightest resource.

Cost

Providers weigh profit vs. affordability — charge by data location, balance concurrent loads.

Optimization Metrics: ChatCache

Caching at the edge



The design of ChatCache: a hierarchical edge cache

A scalable edge system with a hierarchical cache serving both single and multiple users — feature extraction, similarity match, and cache lookup intercept requests before they reach the cloud.

- *Metrics interact — energy limits can force a load to a layer where latency suffers, so weight and prioritize per workload.*

> 91.7%

lower user-perceived latency for voice requests

> 81.6%

lower latency for text requests

Lanyu Xu, Arun Iyengar, and Weisong Shi. “ChatCache: A hierarchical semantic redundancy cache system for conversational services at edge”. In: 2021 IEEE 14th International Conference on Cloud Computing (CLOUD). IEEE. 2021, pp. 85–95.

Technical Challenges in Edge Computing

Challenge	Description
Programmability	Developing & deploying apps on heterogeneous edge nodes needs new programming models.
Program partitioning	Split apps across edge nodes, weighing resources, energy consumption and response latency.
Naming conventions	Efficient, standardized naming for dynamic, heterogeneous edge environments.
Data abstraction	Preprocess at the gateway; handle format diversity, abstraction level and reliability.
Scheduling strategies	Optimize resource use, cut latency and boost task processing in heterogeneous settings.
Offloading & load balancing	Distribute data & tasks across devices to prevent overload, cut latency and add reliability.
Optimization metrics	Use latency, bandwidth, energy and cost to optimize load across edge and cloud layers.

Summary

- New models — Firework, OpenStack++, FocusStack, SpanEdge plus PDG-driven partitioning.
- DNS/IP don’t fit; NDN, MobilityFirst & edgeOS help. Data abstraction has three open challenges.
- Schedule by graph mapping & delay, offload fully or partially, balance load with AI + SDN.
- Latency, bandwidth, energy and cost interact — weight them per workload.

Practice

1. How does edge caching contribute to reducing latency in edge computing?

2. Compare NDN, MobilityFirst and edgeOS naming — when is each a fit?

3. How would you weight latency, bandwidth, energy and cost for a given workload?

Project

Explore resource allocation strategies in edge-cloud environments and implement a resource allocation mechanism that dynamically balances workloads between edge devices and the cloud, based on real-time conditions like network latency and device capabilities. Use simulation tools such as CloudSim (https://github.com/Cloudslab/cloudsim) or iFogSim (https://github.com/Cloudslab/iFogSim) to model the edge-cloud environment and implement the resource allocation mechanism. Evaluate the mechanism's effectiveness by running simulations with varying network conditions, task complexities, and resource constraints.

➢ Tasks

1. **Model the environment** Set up an edge–cloud topology in CloudSim or iFogSim — devices, brokers, cloud datacenter.
2. **Design the policy** Write an allocation rule that routes each task by live latency, device load and capability.
3. **Balance dynamically** Offload to the cloud when the edge is saturated; keep latency-critical tasks local.
4. **Run & evaluate** Sweep network conditions, task complexity and resource limits; measure latency, energy & utilization.

➢ Measures

- **Latency** End-to-end response time as network delay and task size vary.
- **Energy** Device vs. cloud energy under each allocation decision.
- **Utilization & balance** How evenly load spreads — and whether any node overloads.
- **Adaptivity** How well the policy tracks changing real-time conditions.

Example: in iFogSim, route face-detection frames to the fog when latency < 50 ms, else to the cloud; compare vs. cloud-only. **CloudSim · iFogSim (github.com/Cloudslab)**

References

- Quan Zhang et al. "Firework: Big data sharing and processing in collaborative edge environment". In: 2016 Fourth IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb). IEEE. 2016, pp. 20–25.
- Kiryong Ha and Mahadev Satyanarayanan. OpenStack++ for cloudlet deployment. Technical report CMU-CS-15-123. School of Computer Science, Carnegie Mellon University, 2015.
- Brian Amento et al. "FocusStack: Orchestrating edge clouds using location-based focus of attention". In: 2016 IEEE/ACM Symposium on Edge Computing (SEC). IEEE. 2016, pp. 179–191.
- Hooman Peiro Sajjad et al. "SpanEdge: Towards unifying stream processing over central and near-the-edge data centers". In: 2016 IEEE/ACM Symposium on Edge Computing (SEC). IEEE. 2016, pp. 168–178.
- Weisong Shi et al. “Edge computing: Vision and challenges”. In: IEEE Internet of Things Journal 3. 5 (2016), pp. 637–646.
- Quyuan Luo et al. “Minimizing the delay and cost of computation offloading for vehicular edge computing”. In: IEEE Transactions on Services Computing 15. 5 (2021), pp. 2897–2909.
- Lanyu Xu et al.. “ChatCache: A hierarchical semantic redundancy cache system for conversational services at edge”. In: 2021 IEEE 14th International Conference on Cloud Computing (CLOUD). IEEE. 2021, pp. 85–95.