FRANKFURT UNIVERSITY OF APPLIED SCIENCES · CLOUD COMPUTING SS 2026 · PROF. DR. CHRISTIAN BAUN

Edge-Computing Monitoring on a Raspberry Pi Cluster

Detecting fire, smoke and people at the edge — and being honest about what the high-availability design really survives

Daniele Amore · Michael Engel · Sandra Gabriel · Stefan Nguyen · Rizki Rahman · Lars Reul · Sofoniya Yohanes · Sebastian Urmann

Ten tasks • one cluster
hatefullate.github.io/CloudComputing

9	36	8	3.02	29,760	0.772	10
PI NODES	CPU CORES	DISKLESS WORKERS	GFLOPS PEAK	TRAINING IMAGES	MAP@50	ALERT RULES

The system, end to end

Hardware nodes run as host services — they are bound to physical devices. Everything else is a Kubernetes workload on k3s.

A camera streams MJPEG to a Hailo-accelerated inference node running a **YOLOv8n model we trained ourselves** on fire, person and smoke.

Every detection is POSTed to a FastAPI backend, which splits it in two: the evidence image into **erasure-coded MinIO**, the queryable metadata into **replicated PostgreSQL**.

Threats fan out to Telegram; every event is pushed live to the browser over MQTT.

— detection & alert
— live push (MQTT)
— stream, storage, page

HARDWARE — HOST SERVICES ON THE PIs

Sensor node
Pi 4 + AI camera
MJPEG stream on :8080
TASK 1

Inference node
Pi 5 + Hailo-8L AI HAT+
YOLOv8n • 3 classes
TASK 6

Bound to physical devices, so they run as host services — not as Kubernetes workloads. Eight Pi 3 workers netboot diskless from the Pi 5 over PXE, TFTP and NFS.

Model trained in-house on 29,760 images.

KUBERNETES (k3s) · NAMESPACE edge-monitoring

Backend
FastAPI · 3 replicas
Active / Active
TASK 7

Mosquitto
MQTT • events/new
pushed over WebSocket

Frontend
React → nginx
live log, REST fallback
TASK 8

PostgreSQL
CloudNativePG
1 primary + 2 standbys
event metadata

MinIO
distributed, 4 nodes
erasure coding EC:2
evidence images

Telegram notifier
fires only on THREAT
delivers, never decides
TASK 9

PEOPLE

Browser
live event log, evidence images, and a Live / Polling badge

Telegram chat
threat alerts, plus the ten Grafana alert rules built for Task 5

Ten tasks

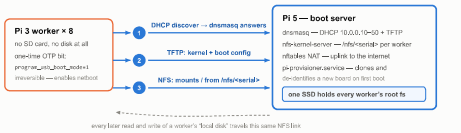
Read down each column. Every number was measured on the physical cluster.

01 Infrastructure

PXE-Boot/

ASKED Set up single-board sensor nodes, consolidate the Pi 3 OS images onto the Pi 5, and enable PXE boot over LAN.

Instead of one SD card per worker, **eight Pi 3s netboot with no storage of their own.** A `pi-provisioner` unit watches the dnsmasq journal and clones plus **de-identifies** the base image by itself, so a bare Pi can be racked and booted with zero manual setup.



CONSEQUENCE — THE THREAD TASK 10 PULLS

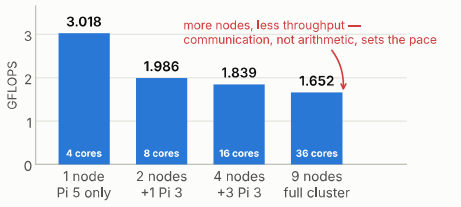
Because the workers own no disk, everything they call "local storage" is a directory on the Pi 5's single SSD.

02 Performance benchmarking

HPL/

ASKED Measure the cluster's GFLOPS capacity with HPL, the benchmark behind the TOP500 list.

The cluster is air-gapped, so nothing was installed per node: `xhpl`, OpenBLAS and OpenMPI 5 live in one shared NFS directory. `N = 4096` was tuned to the Pi 3's 1 GB ceiling; every residual check PASSED.



FINDING

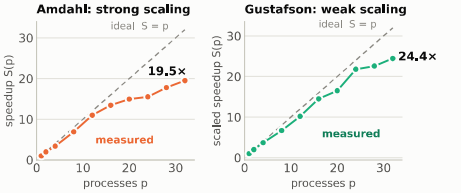
At fixed problem size, **throughput falls as nodes are added.** Each core finishes its slice in milliseconds, so nearly all of the 27.75 s is network exchange — and the Pi 3s speak 100 Mbit Fast Ethernet against the Pi 5's Gigabit. Synchronous MPI decomposition is the wrong shape for this hardware.

03 MPI & scalability

MPI/task_3/

ASKED Deploy MPI with the Pi 3s as workers; demonstrate both scaling laws with at least two MPI applications.

MPI (Message Passing Interface, here OpenMPI) is a standardized protocol for exchanging data across distributed-memory systems via explicit messages between processes. Here, each worker gets exactly four slots, so `p = 4, 8, 12 …` each add **one whole physical node.**



Monte Carlo π estimation, one `MPI_Reduce` per run.

BOTTLENECKS

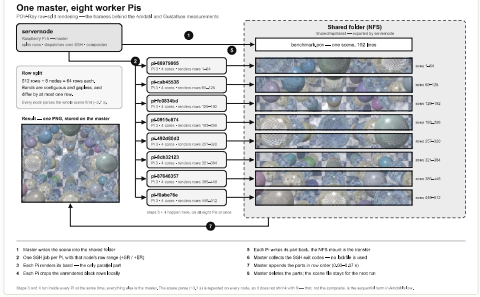
A second example (matrix–vector multiplication) hit two real bottlenecks: a **shared memory bus** within one Pi 3 (compute time barely moves from `p = 1` to `p = 4`), then **100 Mbit/s Ethernet** once multiple nodes join in, unlike the near-ideal π run above.

04 Scaling laws without MPI

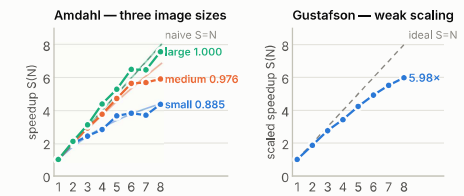
PovRay/ — "pi-scaling-lab"

ASKED Demonstrate Amdahl's and Gustafson's Laws using a non-MPI tool such as Task Distributor.

One harness, `povbench.py` — pure Python standard library, no MPI and no dependencies on the workers. It splits a POV-Ray render into horizontal bands of rows, one SSH job per Pi, and the shared NFS folder is the transfer.



512 rows + 8 nodes = 64 rows each. Task Distributor yields one wall-clock number per run, leaving `P` to be fitted afterwards; this harness **times each phase separately**, so the sequential fraction is measured and can be checked against the fitted one.



Median of 3 runs per point, fitted `P` beside each series. Run order randomised, ≥30 s cooldown, `performance` governor pinned, throttle state logged — each rule added because of a failure mode seen on this cluster.

BOTH LAWS, MEASURED

Amdahl: the fitted parallel fraction rises with problem size, moving the ceiling from 8.7× to 41.1× — the limit belongs to the workload, not the cluster. **Gustafson:** time per job rose 33.7 %, and load imbalance (+14.59 s) plus composite growth (+9.56 s) account for 24.15 s of the 24.18 s deviation.

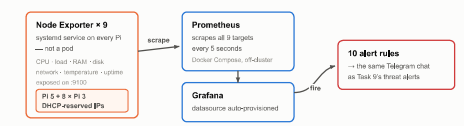
WHAT PER-PHASE TIMING REVEALED

The master's composite costs 0.03–0.07 s — it is **not** the sequential bottleneck everyone assumes. The real `(1-P)` term is the **~3.7 s scene parse every node repeats** before rendering a single row.

05 System monitoring

MonitoringPIs/

ASKED Deploy a monitoring solution that observes hardware health — OS parameters, services and network.



TWO RULES FIRE WITH ZERO DELAY

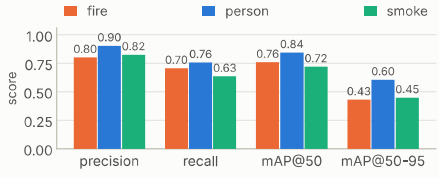
ServerNode down and **NFS root read-only** — the Pi 5 is the control plane and the root filesystem of every diskless worker, so neither is a generic node-down case. The other eight carry a 1–5 minute pending window.

06 Object detection model

ai-model/

ASKED Collect enough images to train and test a detector. Training your own is required — a pre-existing model is explicitly not sufficient.

A YOLOv8n trained from scratch on a **self-assembled 29,760-image** dataset: 300 epochs max, patience 50, 640 px input. Best weights export to ONNX and compile for the **Hailo-8L** accelerator. Overall: precision 0.843, recall 0.695, mAP@50 0.772, mAP@50-95 0.493.



FINDING

Smoke is the weakest class (recall 0.633) — 546 smoke instances in training against 105,452 person, roughly **193 : 1.** The mAP@50 / mAP@50-95 gap says it finds objects but does not always box them tightly.

07 Backend & storage

backend/ · cluster/

ASKED Build a backend managing the sensor nodes and their data, containerised on a k3s Pi cluster with a distributed file system or MinIO.

Backend	3 stateless replicas	Active/Active
PostgreSQL	CNPG, 1 primary + 2 standbys	Active/Passive
MinIO	4 nodes, erasure coding EC:2	Active/Active

Active/Active = all replicas serve; **Active/Passive** = one serves, a standby is promoted (~5 s here).

Two stores, not one: MinIO holds the binary evidence images, PostgreSQL the structured metadata — "the 50 newest THREAT events by time" is a query an object store cannot answer natively.

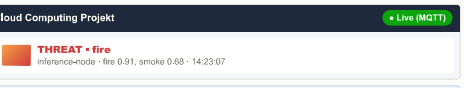
SHARED NOTHING IN KUBERNETES, SHARED DISK IN REALITY

476 MiB allocatable per Pi 3 forces Postgres to a 192Mi request — at 256Mi it and MinIO will not fit one node. And the workers are diskless: all four MinIO fragments and all three Postgres copies are directories on **the one SSD in the Pi 5.** Redundancy spans nodes, not devices.

08 Frontend

frontend/

ASKED Present log information and event messages in a Vue or React frontend on k3s, over REST or MQTT.



REST STAYS THE SOURCE OF TRUTH

React/Vite in a two-stage image — Node builds, nginx serves. Events load once over REST, then arrive on the `events/new` MQTT topic; if the socket drops, the UI **falls back to a 5-second REST poll.** Both paths are same-origin through Traefik — no CORS anywhere.

09 Telegram notifications

telegram-bot/

ASKED Implement a Telegram bot informing team members about infrastructure health events and detected threats.

IT DELIVERS; THE BACKEND DECIDES

Keeping classification in the backend leaves the notifier reusable — which is why Task 5's ten Grafana rules point at the same chat. Members subscribe with `/start`; with no bot token it runs **dry-run**, so CI can test it.

10 · Documentation and the risk register

docs/ · pages/ — complete online documentation via GitHub Pages, plus the document that argues against our own design.

ASKED Create complete online documentation via GitHub Pages and present it with a poster and a live demonstration — no slide decks, no PDF report.

A multi-page site organised **by task rather than by directory**: architecture, a storage design document, a build runbook per migration step, and an incident log of every problem hit moving onto the physical Pis. Alongside it, `docs/weaknesses.md` — a risk register written **deliberately before the demo, not after**: failure mode → blast radius → why the mitigation does not cover it → where the fix is tracked.

ALSO REPORTED, NOT SWEPT UP

The worker NFS exports use `sec=sys` with a wildcard host list — required for diskless boot, but wider than necessary. MinIO's readiness probe returns 200 **even with zero usable drives**, and a backend liveness probe was once seen **killing a healthy pod.**

Explicitly not verified: permanent node-loss recovery, bucket versioning, behaviour under sustained load, any capacity claim.

WHAT KUBERNETES REPORTS

Replicas spread across seven distinct nodes.

minio-0	minio-1	minio-2	minio-3
shard 1/4	shard 2/4	shard 3/4	shard 4/4
pi-worker-1	pi-worker-2	pi-worker-3	pi-worker-4

pg-1 • PRIMARY	pg-2 • standby	pg-3 • standby
writes	WAL stream	WAL stream
pi-worker-5	pi-worker-6	pi-worker-7

Verified on hardware, 2026-08-14: a real node power-off survived; a standby auto-promoted with a single 5-second gap.

WHAT PHYSICALLY EXISTS

The workers are diskless — every "local" volume is an NFS path.

Every replica, every copy, every root filesystem
4 MinIO erasure shards · 3 PostgreSQL copies
8 Pi 3 worker root filesystems · every PersistentVolume
k3s local-path provisioner → a directory on "the node's local disk"

one SSD in the Pi 5
reached by every worker over NFS

Survives a worker loss — not the loss of this device.

Kubernetes is not lying — the redundancy genuinely survives a worker crashing, rebooting or being swapped for an identical board. What it does not survive is the loss of the one device underneath all of it.

VERIFIED ON THE PHYSICAL CLUSTER

4 MinIO pods, 4 distinct workers	● 200
Quorum split, 3 of 4 drives	● read + write
Quorum split, 2 of 4 drives	● write fails, correctly
CloudNativePG failover	● ~5 s, no data loss
Real node power-off	● writes kept working
`O_DIRECT` over the NFS mount	● supported

SINGLE POINTS OF FAILURE, AS INVENTORIED

● Backend • PostgreSQL • MinIO	redundant
● Storage volumes	logically spread only
● The Pi 5's SSD	total data loss
● Control plane + ingress	single node
● Off-node backups	none yet
● PXE / DHCP / TFTP / NFS	single service

CONCLUSION

Ten tasks on one nine-node Raspberry Pi cluster: both scaling laws demonstrated twice — with MPI and without — a detection model trained from scratch rather than downloaded, and a high-availability stack whose limits were measured on real hardware and then written down.

FULL DOCUMENTATION
hatefullate.github.io/CloudComputing
github.com/Hatefullate/CloudComputing