OPC UA Industrial Plant HUB

A Distributed Data Collection System for Industrial Automation

Federico Capitanio
federico.capitanio@studio.unibo.it

Distributed Systems Course - A.A. 2025-2026
University of Bologna

The Problem

Industrial Data Collection Challenges

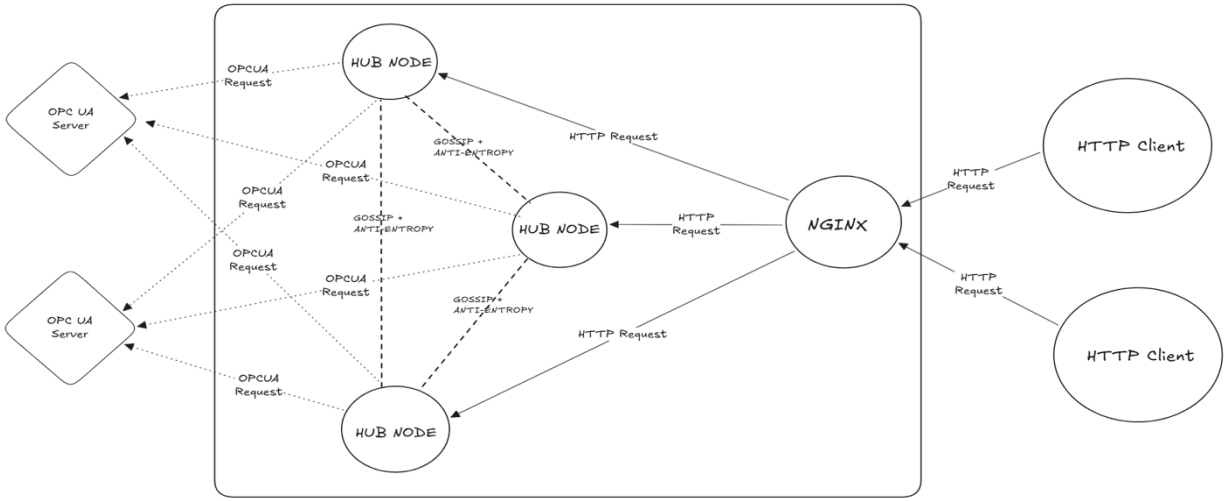
- **24/7 Operations**
  Plants require continuous monitoring
- **Single Point of Failure**
  Traditional architectures are risky
- **Geographic Distribution**
  Plants span multiple locations
- **Dynamic Reconfiguration**
  Add sources without downtime

Distributed Multi-HUB Solution

- **3+ HUB Nodes**
  Parallel OPC UA data collection
- **No Central Coordinator**
  Peer-to-peer synchronization
- **nginx Load Balancer**
  Client transparency + health checks
- **Dynamic Addition**
  Runtime OPC UA server config

System Architecture

Hybrid Client-Server / Peer-to-Peer Design



Key Components:

Client-Server Layer: REST API with location transparency

P2P Layer: Gossip + Anti-Entropy protocols

HUB Nodes: OPC UA collection + Local SQLite + REST server

CAP Theorem Positioning

Why AP (Availability + Partition Tolerance)?

Property	Choice	Rationale
Consistency	Eventual	Industrial monitoring tolerates 30s lag
Availability	✅ **High**	24/7 operations cannot tolerate downtime
Partition Tolerance	✅ **Full**	Nodes must operate independently during network failures

Design Decision:

Clients prefer **"stale data"** over **"no data"**

Strong consistency → synchronous coordination → higher latency + lower availability

Eventual consistency with convergence guarantee: **<60 seconds**

Gossip Protocol

Heartbeat-Based Distributed Failure Detection

Protocol Flow:

1. Every 5 seconds
Each HUB → POST /internal/gossip/ping to peers

2. Message includes Lamport Clock
(sender's logical time)

3. Peer responds
ACK + its own LC

4. API Timeout 3s
→ node marked as not responsive

5. 3 missed heartbeats (15s)
→ node marked DEAD

Properties

- **Strong Completeness**
  All survivors detect failure
- **Weak Accuracy**
  False positives possible, automatic recovery

Validation

Detection time measured:

15.6s ± 0.4s

(5 test runs)

Anti-Entropy Protocol

Pull-Based Eventual Consistency

Synchronization Flow:

1. Every 10 seconds
HUB check local Lamport Clock state for each peer

2. Request sync from alive peers
POST /internal/anti-entropy/sync with LC state vector

3. Peer identifies gaps
`Queries local data not yet propagated to requester based on LC comparison and respecting casual ordering)`

4. Returns missing data points
Delta containing [node_id, LC, tag, value, …]

5. Requester converges state
Merge received data points
UpdateLC = max(local_lc, received_lc) per node
Conflict resolution: Last-Write-Wins

Why Pull-Based?

✓ Natural backpressure
(receiver controls rate)

✓ No tracking overhead
(sender doesn't track)

✓ Partition resilient
(auto-resumes)

Validation

Convergence time:

10-25s
(typical 15s)

Lamport Clocks

Casual Ordering Without Synchronized Time

Implementation:

- Each HUB maintains **local LC** (integer counter)

- Increment on every event:
  - Data point collected → LC++
  - Server config added → LC++
  - Message sent/received →

```
LC = max(local, received) + 1
```

- Data stored with metadata:
  (node_id, Lamport_clock, tag, value, …)

Conflict Resolution: Last-Write-Wins

Example:

- Node A adds server config
  → assigns **LC = 1000**

- Node B adds same server
  (during partition)
  → assigns **LC = 1001**

After sync:
Conflict resolved by highest LC
→ Node B config “wins” (deterministic)

Key Property: Provides casual ordering across distributed events without physical clock sync.

Key Design Decisions

1. Leaderless Multi-Master

- Every node = source + replica
- No master/coordinator
- No SPOF at application layer

2. Local Storage + Replication

- Each HUB: SQLite database
- No central DB dependency
- Trade-off: Eventual consistency

3. Stateless HTTP API Layer

- JWT-based authentication
- Horizontal scalability
- nginx round-robin distribution

4. Dynamic Configuration

- Add servers via POST API
- Config propagates <30s
- No system restart required

Technology Stack

Backend

- Python 3.12+
- FastAPI + uvicorn
- SQLite (local DB)
- asyncua (OPC UA)

Infrastructure

- Docker Compose
- nginx (TLS + LB)
- TLS 1.2/1.3 encryption
- Nodes health checks

Distributed Protocols

- Gossip Protocol
- Anti-Entropy
- Lamport Clocks
- JWT Auth

Testing Strategy

3-Level Validation Approach

1. Functional Testing

- OPC UA connection + polling
- Anti-Entropy data synchronization
- Dynamic server addition
- Gossip failure detection
- Nginx traffic redirection
- JWT auth + RBAC

2. Non-Functional Testing

- Eventual consistency
- Network partition (iptables)

3. Deployment Testing

- Docker Compose from scratch
- Persistence across restarts

Performance Results

Gossip Failure Detection

Target: Detect within 15s

Measured: 15.6s ± 0.4s

✅ **PASSED**

Anti-Entropy Convergence

Target: Converge <60s

Measured: 10-25s (typically 15s)

✅ **PASSED**

Load Balancer Failover

Target: Redirect <30s

Measured: 0 errors, seamless

✅ **PASSED**

Dynamic Server Addition

Target: Propagate <30s

Measured: Node-B: 12s, C: 18s

✅ **PASSED**

Limitations & Future Updates

1. **OPC UA Full Tree Browsing** • Startup 5-10 min with >10K nodes
   Future: Selective and dynamic OPC UA node configuration

2. **SQLite Scalability** • ~200 writes/s single-writer limit – possible constant lock contention
   Future: Migrate to PostgreSQL/TimescaleDB/InfluxDB

3. **Stateless JWT** • No immediate revocation (24h validity)
   Future: Token blacklist in Redis

4. **Single nginx Instance** • Infrastructure SPOF
   Future: nginx replication + DNS LB

5. **No Observability** • Missing Prometheus metrics/tracing
   Future: /metrics endpoints + Grafana

Learning Outcomes

Hands-On Experience with Distributed Systems Concepts

- **CAP Theorem**: Practical tradeoffs between C/A/P through AP implementation
- **Logical Clocks**: Lamport clocks for causal ordering without synchronized time
- **Eventual Consistency**: Convergence guarantees with measurable times (10-60s)
- **Failure Detection**: Gossip Protocol with completeness/accuracy analysis
- **Replication Strategies**: Pull-based vs push-based (backpressure advantages)

Key Insight: *Distributed systems principles apply to industrial automation, bridging "theoretical" aspects and real-world manufacturing requirements*

Thanks for the attention

Federico Capitanio
federico.capitanio@studio.unibo.it

Distributed Systems Course - A.A. 2025-2026
University of Bologna