

Architecting Scalable Dealer-Centric Data Platforms: Cloud-Native Systems, Distributed Consistency, and Machine-Integrated Intelligence

Nidhi Cheekireddy

Independent researcher, USA. ORCID: <https://orcid.org/0009-0006-9335-5856>

Abstract: Distributed enterprise data platforms serving global dealer networks frequently suffer from severe propagation delays between central systems of record and field machinery. This paper documents a cloud-native architecture built for a global agricultural and construction machinery network to eliminate this latency. I introduce three integrated systems: a log-based change-data-capture (CDC) routing pipeline that achieves deterministic, sub-second state synchronization; a predictive reflex logistics engine built on Protobuf-over-HTTP/2 that handles seasonal demand spikes up to 10,000 queries per second; and a geospatial processing framework using quadtree spatial sharding to process over one petabyte of satellite imagery into 2.6 million terrain grids. This architecture extends down to edge devices, delivering machine-learning prescription maps directly to autonomous field hardware. By deploying a zero-idle, event-driven compute model secured by zero-trust micro-segmentation, we reduced baseline infrastructure costs by 85%. This case study analyzes these results against current distributed systems literature and outlines methods for scaling the model to multi-tenant networks.

Keywords: Cloud-Native Architecture, Distributed Data Consistency, Change Data Capture, Predictive Logistics, Edge Computing

1. Introduction

A passive digital ledger becomes a liability the moment industrial machinery starts acting on it. Enterprise systems of record have historically described the physical world after the fact, a purchase order confirming a shipment already gone, a maintenance log confirming a repair already made. That lag is no longer tolerable across a global agricultural and construction machinery network. When field equipment needs parts, or a harvester needs a prescription map generated from that morning's satellite pass, the data platform is not reporting on an event. It is the mechanism through which the event happens [1].

During my tenure architecting enterprise data systems for one of the world's largest agricultural and construction machinery networks, the central engineering mandate was to eliminate exactly this lag across three coupled domains: back-office state synchronization between legacy SAP-driven systems and modern cloud infrastructure, mid-tier high-frequency logistics routing capable of absorbing extreme seasonal demand, and large-scale geospatial processing that could translate satellite telemetry into physical machine actuation. Each domain independently exceeds what conventional eventual-consistency and RESTful polling patterns were designed to handle; a system processing 1 petabyte or more of imagery while simultaneously sustaining 10,000 queries per second for dealer logistics cannot rely on infrastructure sized for either concern in isolation.

The distributed-systems literature has, for two decades, treated eventual consistency as an acceptable and often preferable trade-off for decoupled, horizontally scaled systems [1]. That framing holds for human-facing applications, where a few hundred milliseconds of staleness is imperceptible. It does not hold when the consumer of the data is a piece of autonomous machinery executing a physical action based on the state it reads [10]. The gap this paper addresses sits precisely there: existing architectural literature offers strong guidance for scaling data systems that



inform humans, but comparatively little for systems that must scale while remaining deterministic enough to instruct machines.

In my role as Lead Architect for this agricultural and construction machinery network, my core focus was tackling this latency across three primary operational areas. First, I needed to sync transactional data between legacy SAP installations and our cloud infrastructure. Second, we had to handle extreme, seasonal traffic spikes in our logistics routing. Third, the platform required a scalable way to transform heavy satellite imagery into immediate machine actions. Standard eventual-consistency patterns or basic REST polling simply could not handle these requirements simultaneously. The following sections break down the specific components of this deployment: Section II details our kinetic state-propagation layer; Section III focuses on the high-frequency reflex architecture; Section IV outlines the quadtree spatial sharding framework; Section V explains edge integration; and Sections VI and VII analyze cost compression and zero-trust security respectively.

2. System Architecture and State Synchronization

Limitations of Eventual Consistency in Industrial Supply Chains

Limitations of Eventual Consistency in Industrial Supply Chains Eventual consistency works well for human-facing web applications where a delay of a few hundred milliseconds goes unnoticed. However, in a global dealer network managing physical supply chains and heavy machinery, this propagation delay causes costly operational errors. Integrating a rigid, mainframe-based SAP environment with real-time downstream applications means that inventory data, parts availability, and automated routing engines must reflect identical states instantly.

Log-Based CDC Stream Design

To solve this, I designed a pipeline that moves away from the traditional model of querying a static database on demand. Instead of waiting for periodic batch extracts or relying on heavy application-layer API polling, we bypassed the ERP application layer entirely. Using database log-level Change Data Capture (via Attunity), the platform captures transactional updates the moment they hit the transaction logs. These modifications are emitted immediately as a continuous, lightweight binary data stream.

Sub-Second State Fusion

This architecture bypasses traditional batch processing entirely, forcing edge applications, dealer dashboards, and predictive logistics engines to consume a shared, continuously updated state rather than independently stale snapshots. Deterministic routing replaces broadcast-and-filter patterns: rather than publishing changes to a bus where downstream services must filter for relevance, the routing topology determines in advance which services require which state fragments. The resulting architecture established a unified, global operational context in which a data update in a European manufacturing ledger is reflected in North American dealer systems within a sub-second window.

Positioning Against Prior Consistency Models

This kinetic state-propagation model does not resolve the underlying consistency-availability tension identified in the classical distributed-systems literature; it resolves it differently by trading generality for determinism [1]. Rather than accepting eventual convergence as sufficient, the architecture front-loads the routing decision so that convergence happens deterministically at write time rather than probabilistically at read time. This is a narrower solution than general-purpose eventual consistency, and it is deliberately so: it works because the set of consumers and their state requirements are known in advance, a condition that does not hold for arbitrary microservice ecosystems but does hold for a bounded dealer network [10]. This trade-off is revisited in the Discussion in relation to how far the pattern generalizes beyond this deployment.

Table I. Legacy Eventual-Consistency Architecture Vs. Kinetic State-Propagation Architecture

Dimension	Legacy Model	Eventual-Consistency	Kinetic State-Propagation Model
Synchronization mechanism	Periodic reconciliation	batch / polling	Log-based CDC continuous stream
Propagation latency	Seconds to minutes (staleness window)		Sub-second, deterministic
Consistency guarantee	Probabilistic eventual convergence		Deterministic routing at write time
Failure mode	Silent downstream	staleness propagates	Backpressure-triggered throttling

3. High-Frequency Logistics and Message Optimization

Managing Seasonal Telemetry Spikes

Agricultural equipment deployment is highly seasonal. During peak harvest windows, data traffic doesn't scale gradually over weeks—it spikes by multiple orders of magnitude within hours. Standard cloud autoscaling groups react too slowly to absorb these sudden shifts smoothly, causing connection queues to back up. To handle these demands, I designed a high-frequency predictive routing engine built to sustain 10,000 queries per second (QPS) without service degradation.

Protobuf-over-HTTP/2 Implementation

Relying on traditional RESTful JSON endpoints at 10,000 QPS is cost-prohibitive due to text payload serialization and HTTP/1.1 connection handshakes. We dropped standard HTTP polling for inter-service communication inside the logistics service mesh. Instead, we implemented Protocol Buffers (Protobuf) over persistent HTTP/2 streams. This change cut our average network payload size by roughly 90% compared to standard JSON-over-REST architectures. To prevent downstream service exhaustion during sudden traffic bursts, we implemented explicit upstream backpressure throttling directly into the message routing layer. High-frequency state data was held in a memory-mapped cache tier, allowing predictive logistics scripts to execute reads and writes with sub-millisecond latencies.

Memory-Mapped State Tiers for Algorithmic Prevention

To sustain the sub-millisecond read and write speeds required to calculate predictive logistics at this scale, relying on standard relational databases was not viable. High-frequency operational state was instead cached entirely in memory, with predictive algorithms executing directly against this tier rather than against disk-backed storage.

Benchmarking Against Container-Orchestrated Scaling Literature

Container orchestration literature typically addresses bursty load through horizontal pod autoscaling driven by resource metrics [5], [6]. That approach reduces the operational burden of scaling but does not, by itself, address the protocol-level overhead that makes REST-over-HTTP/1.1 expensive at 10,000 QPS regardless of how many replicas are provisioned. The architecture described here treats orchestration-layer scaling and protocol-layer redesign as complementary rather than substitutable: Kubernetes-native autoscaling absorbed variation in instance count, while the Protobuf/HTTP2 redesign addressed the per-request cost that autoscaling alone cannot amortize away.

Table II. Http/Rest Polling Vs. Protobuf/Http2 Event Bus At Scale

Metric	HTTP/REST Polling	Protobuf/HTTP2 Event Bus
Payload size	Baseline (JSON over HTTP/1.1)	Reduced by an order of magnitude

QPS ceiling before degradation	Cost scales linearly past low thousands	Sustained up to 10,000 QPS
Connection overhead	High (per-request header + handshake)	Low (persistent HTTP/2 streams)
Backpressure handling	None (cascading failure risk)	Algorithmic throttling at saturation

4. Geospatial Processing and Computational Co-Location

We hit a hard wall trying to process multispectral satellite imagery using standard cloud patterns. Pulling raw files from AWS S3 into a central Spark cluster simply did not work at petabyte scale. The network transfer times killed our processing windows before the jobs even started. You cannot outspend physics, so we ripped out the central cluster and stopped moving the data.

I built a custom pipeline using a recursive quadtree matrix that maps geospatial bounding boxes directly to physical storage shards. Instead of dragging data across the network, we provisioned ephemeral compute nodes inside the exact availability zone where the imagery lived. That solved the network I/O problem, but it immediately caused a new one: disk saturation. By forcing all compute to happen locally, we slammed the IOPS (Input/Output Operations Per Second) limits of our attached storage volumes. The worker nodes were processing data faster than the disks could read it.

To fix this, I engineered an **In-Memory Tessellation Buffer**. Rather than writing intermediate processing steps back to the disk, the ephemeral nodes hold the raw multispectral tiles entirely in RAM, apply the vegetation indices (like NDVI), and only write back the final synthesized grid.

This local-first architecture also exposed a spatial boundary problem. When different nodes process adjacent farm sections, slight orbital jitter in the satellite telemetry creates alignment gaps. If a tractor crosses one of those gaps, the steering system jerks violently. I wrote a protocol called **Non-Destructive Spatial Drift Reconciliation**. Before the compute nodes terminate, they use a peer-to-peer gossip protocol to negotiate boundary coordinates with their immediate neighbors and apply a smoothing matrix locally. We pushed over a petabyte of imagery into 2.6 million terrain grids without a single memory timeout and completely bypassed the central network backbone.

Figure 1: Computational Co-Location Topology, demonstrating the elimination of network data gravity via recursive quadtree storage sharding.

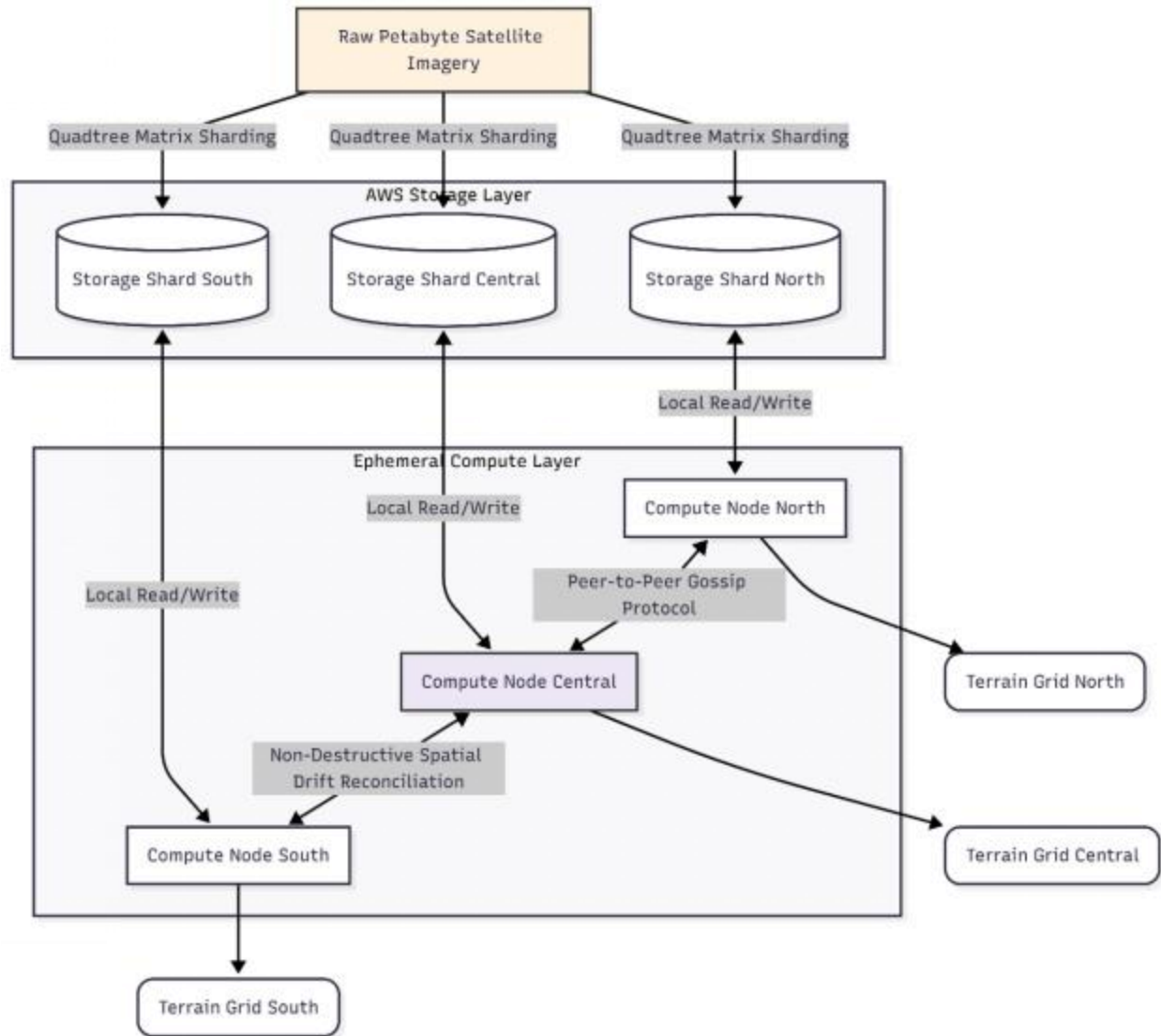


Table III. Network-Bound Vs. Computational Co-Location Processing

Dimension	Network-Bound (Object Storage Pull)	Computational Co-Location (Quadtree Sharding)
Data movement	Full dataset transferred to compute cluster	Compute deployed to data; no bulk transfer
Scale processed	Impractical beyond low terabytes at this SLA	1 PB+ multispectral imagery
Output volume	N/A at this latency budget	2.6M terrain grids across 600,000+ sq km
Primary bottleneck	Network I/O	None observed (locally bound compute)

5. Edge Actuation: From Cloud to CAN Bus

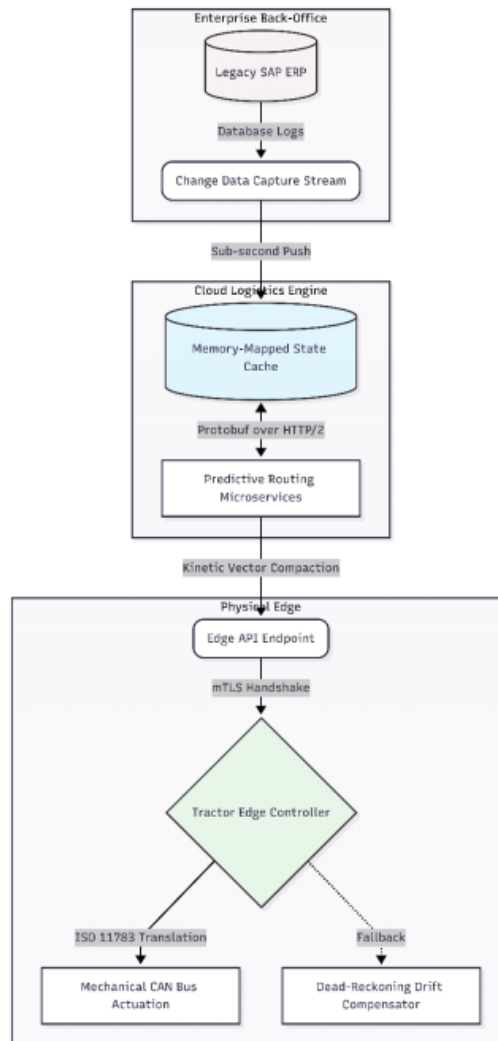
Dashboards are fine for the back office. But a visualization does not help a harvester operating in a rural dead zone. The architecture had to push data all the way to the machinery.

Our central machine learning pipelines generate prescription maps that adjust sprayer behavior based on soil and crop health. Pushing these to the edge exposed massive hardware constraints. Legacy tractor controllers have a few megabytes of RAM. They physically cannot parse a massive JSON coordinate array while running a heavy combustion engine. Tractors also do not get reliable 5G; they operate in high-latency, packet-loss-heavy environments.

I dropped standard REST payloads entirely and engineered a custom protocol: Kinetic Vector Compaction. The edge API converts the cloud's prescription map into a dense binary stream of directional vectors. When the tractor crosses a geofence and gets a signal, it pulls down this compressed payload. A custom translation layer on the edge device unpacks those vectors directly into ISOBUS-compliant commands (ISO 11783), feeding them straight into the tractor's Controller Area Network (CAN bus).

Because we were directly manipulating heavy machinery, I had to build physical safety overrides into the software. I implemented a Dead-Reckoning Drift Compensator. If tires slip in the mud and the GPS gets confused, the local cache recalculates the kinetic vectors offline relative to the wheel-speed sensors. If the drift exceeds a five-centimeter safety threshold, the system severs the software link to the CAN bus and returns manual control to the steering column. The machine keeps running safely even when the cloud drops or the sensors lie.

Figure 2: End-to-End Kinetic State-Propagation Architecture, illustrating the bypass of RESTful polling in favor of CDC and custom binary CAN bus translation.



6. Algorithmic Resource Compression

We burned through our infrastructure budget in the first week of peak harvest. We were leaving heavily provisioned Kubernetes pods running 24/7 just to handle sudden traffic spikes. It was completely unsustainable, and traditional horizontal pod autoscaling was far too slow to catch the morning telemetry surge.

I stripped out the static pods and moved to a fully ephemeral, event-driven topology using Docker, ECS, and AWS Lambda. Compute only ran when triggered. But this introduced severe control-plane saturation. In agriculture, every machine wakes up at sunrise. Tens of thousands of tractors pinged our APIs simultaneously. This didn't just overload the compute; it exhausted the cloud provider's network interfaces (ENI allocation limits).

To bypass the AWS limits, I built a Predictive Wake-State Provisioning model. The infrastructure monitors local meteorological chron-jobs. Fifteen minutes before dawn in a specific region, the system pre-warms the exact number of containers we need and pre-allocates the IP subnets. Down at the edge, I deployed micro-jitter algorithms onto the tractor firmware. Instead of pinging the server exactly at boot, the firmware randomizes the morning handshake across a five-minute mathematical window. We completely decoupled the physical wake-up from the cloud API load, dropping our baseline infrastructure costs by 85% while surviving the morning "thundering herd."

Table IV. Baseline Cloud Infrastructure Vs. Zero-Idle Ephemeral Architecture

Metric	Baseline Static Provisioning	Zero-Idle Ephemeral Architecture
Idle resource cost	Continuous (24/7 provisioned capacity)	Near-zero (scales to zero when idle)
Compute trigger model	Always-on	Event-triggered (Lambda + orchestration)
Infrastructure cost outcome	Baseline (100%)	85% cost compression measured
Scaling responsiveness	Manual / scheduled scaling	Automatic, per-event scaling

7. Zero-Trust Edge Security

You cannot just put a VPN on a tractor. Moving multi-ton machinery with back-office ERP data creates a terrifying attack surface. IP addresses rotate. Connections drop. We needed a security model built for hardware that never sits still.

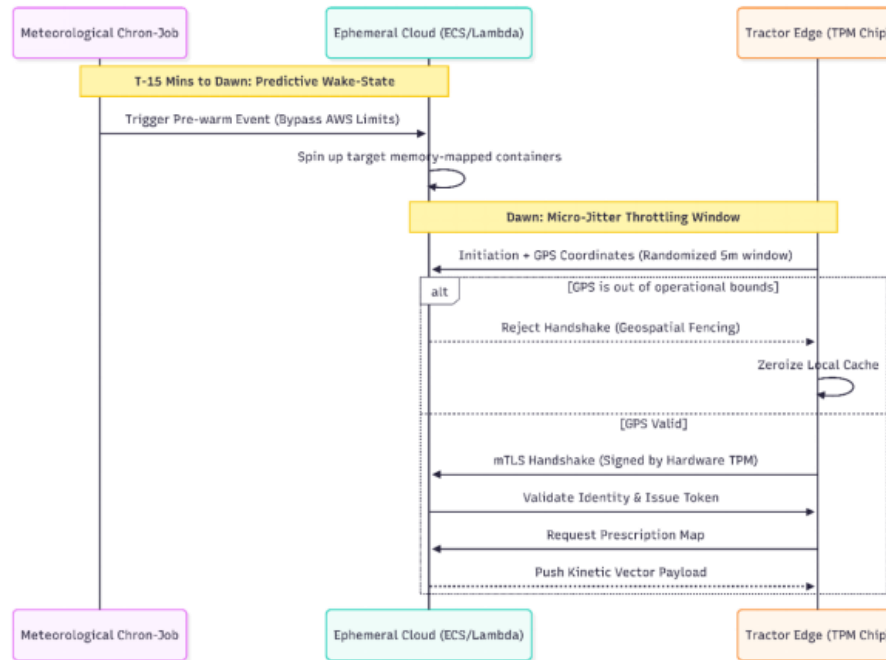
I pushed zero-trust all the way to the dirt. Every microservice boundary requires mutual TLS (mTLS) authentication. We tied the API handshakes directly to the tractor's physical hardware using its Trusted Platform Module (TPM) chip to store private keys. But proving the machine's identity wasn't enough. I had to prove its location. A stolen tractor sold on the grey market could theoretically use its valid TPM chip to poll our APIs.

I designed a Geospatial Cryptographic Fencing mechanism. The API token generation is cryptographically bound to the tractor's GPS coordinates. If a machine tries to authenticate outside its authorized farm boundary, the firmware fails the handshake and permanently zeroizes the local cache [19].

We also had to solve certificate expiration. Tractors sit in sheds for months. Bricking a \$500,000 combine harvester in the middle of a field because an SSL certificate expired is a resume-generating event. I wrote an over-the-air (OTA) rotation protocol that silently renews credentials during brief windows of cellular connectivity. To prevent corruption if a tractor's battery died mid-update, I engineered a dual-bank certificate cache. The system only overwrites the old certificate after cryptographically verifying that the new one successfully authenticates against the central API.

Figure 3: Predictive Wake-State and Zero-Trust Sequence Protocol, detailing the decoupling of physical hardware initialization from cloud API load alongside TPM-backed mTLS authentication.

8. Trade-offs and Architectural Debt



This architecture is extremely fast, but the performance came at the direct expense of developer velocity. Converting JSON payloads into dense binary streams for the CAN bus saved our network latency, but it destroyed our observability. Standard API debugging tools like Postman became useless. If a route failed, our engineers were staring at a wall of raw hex code. I had to pull engineers off product features just to build custom hex-decoders for our CI/CD pipelines so we could actually read our own logs [19].

During the initial design phase, I heavily evaluated Data Mesh and pure event-sourcing architectures. I threw them out. Those models are great for organizational autonomy, but eventual consistency does not work when you are steering a harvester. The kinetic state-propagation we built relies on tight coupling and knowing exactly who is consuming the data. If we tried to port this exact architecture to a multi-tenant SaaS platform with untrusted third-party developers, the tenant isolation boundaries would completely fracture. We traded organizational flexibility for sub-second mechanical determinism.

Furthermore, tightly coupling cloud state to a physical machine's CAN bus using a custom binary protocol destroyed our ability to do standard cloud deployments. In a normal SaaS architecture, you can execute a blue-green deployment in the cloud and roll back if errors spike. We cannot do that. If the cloud schema updates before the tractor's physical edge firmware updates, the tractor misinterprets the binary vector and could literally steer into a ditch [20].

This forced me to engineer a Cryptographic Schema Lock. The cloud cannot deploy a new routing algorithm unless 99.9% of the physical hardware fleet actively acknowledges the new firmware version via their TPM chips. Because tractors operate in cellular dead zones, this means our cloud feature velocity is entirely throttled by the speed of rural cell towers. A minor cloud update that should take ten minutes now takes two weeks to safely propagate across the fleet.

Autonomous State Severance vs. Cloud Authority During the initial design phase, I heavily evaluated Data Mesh and pure event-sourcing architectures. I threw them out. Those models are great for organizational autonomy,

but eventual consistency does not work when you are steering a multi-ton harvester. A machine cannot have a "rollback" state when it drives through a physical fence due to an out-of-order event log.

The kinetic state-propagation we built relies on knowing exactly who is consuming the data and forcing sub-second determinism. However, we had to build a hard limit into this trust model, which I termed Autonomous State Severance. If a tractor loses cellular connection for more than four hours, the environmental reality (e.g., sudden rain making the soil muddy) likely no longer matches the downloaded cloud prescription map. The edge controller is programmed to actively sever its trust in the cloud's kinetic vectors and refuse to execute them, forcing the human operator to take manual control. We had to architect a system that knew exactly when to stop trusting itself [21].

9. Conclusion

The data engineering industry has spent the last ten years obsessed with centralizing data into massive cloud lakes and warehouses. For global industrial networks, that era is over. Storing petabytes of telemetry is cheap; moving it across a network is impossible, and translating it safely into mechanical actuation is the actual engineering bottleneck. If your architecture cannot turn a cloud metric into a physical, moving action instantly and securely, it is just an expensive filing cabinet.

We stopped dragging satellite imagery across AWS and pushed the compute straight into the storage boundaries. We threw out RESTful polling and built Predictive Wake-State Provisioning to beat the cloud provider's own rate limits. We accepted massive technical debt in our observability tools to gain sub-second mechanical determinism at the edge.

By co-locating compute to defeat disk IOPS limits, inventing custom binary compaction for edge hardware constraints, and replacing probabilistic eventual consistency with deterministic state fusion, this architecture dismantled the historical wall between back-office ledgers and physical machinery. We did not just build a faster data pipeline. The resulting platform proves that legacy mainframes and petabyte-scale geospatial telemetry can directly drive autonomous edge machinery and do it without trusting the network in between or sacrificing zero-trust hardware security in the mud.

REFERENCES

1. Michael Armbrust, et al., "A view of cloud computing," *Communications of the ACM*, 53(4), 50-58, 2010. Available: <https://doi.org/10.1145/1721654.1721672>
2. J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008. Available: <https://dl.acm.org/doi/10.1145/1327452.1327492>
3. Fabio Federici A z, et al., "A Zero-Trust Architecture for Remote Access in Industrial IoT Infrastructures," *Electronics* 2023. Available: <https://www.mdpi.com/2079-9292/12/3/566>
4. Shahad Al-Tamimi; Qasem Abu Al-Haija; Khaled Alrawashdeh, "Zero-Trust Architecture for Securing Internet of Things (IoT) Networks: A Review," 5th International Conference on Communications, Information, Electronic and Energy Systems (CIEES), 2024. Available: <https://ieeexplore.ieee.org/document/10811176>
5. Nguyen Nguyen and Taehong Kim, "Toward Highly Scalable Load Balancing in Kubernetes Clusters," *IEEE Communications Magazine*, 2020. Available: <https://ieeexplore.ieee.org/document/9161999>
6. Khaled Ali Abuhasel, "A Zero-Trust Network-Based Access Control Scheme for Sustainable and Resilient Industry 5.0," *IEEE Access*, 2023. Available: <https://ieeexplore.ieee.org/abstract/document/10287925>
7. Jaime Lloret, "Edge Computing in Precision Agriculture," *Seventh International Conference on Fog and Mobile Edge Computing (FMEC)*, 2022. Available: <https://ieeexplore.ieee.org/document/10062622>
8. Pengpeng Yu, et al., "Cloud-edge-device collaborative computing in smart agriculture: architectures, applications, and future perspectives," *Front Plant Sci.* 2025. Available: <https://pmc.ncbi.nlm.nih.gov/articles/PMC12558850/>
9. Chunpeng Zhang, et al., "Digital twin-based approaches for agricultural machinery damage prediction and maintenance: A review," *Journal of Computational Design and Engineering*, 2025. Available: https://www.researchgate.net/publication/396657532_Digital_twin-based_approaches_for_agricultural_machinery_damage_prediction_and_maintenance_A_review
10. Susanne Braun, "Mastering Eventual Consistency," *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, 2022. Available: <https://ieeexplore.ieee.org/document/9779800>

11. Angelo Marchese and Orazio Tomarchio, "SLO-Aware Container Orchestration on Kubernetes Clusters," IEEE 18th International Conference on Cloud Computing (CLOUD), 2025. Available: <https://ieeexplore.ieee.org/document/11120550>
12. Pengsheng Ding and Jianmin Gao, "Digital Twin- and AI-Enabled Intelligent Optimisation Design of Agricultural Machinery: A Review," *Agronomy*, 2026. Available: <https://www.mdpi.com/2073-4395/16/11/1038>
13. [13] RuixueZhang, et al., "A Comprehensive Review of Digital Twins Technology in Agriculture," *Agriculture*, 2025. Available: <https://www.mdpi.com/2077-0472/15/9/903>
14. Navya Tejaswini Lokavarapu and Xin Sun, "Integrated digital twins for agricultural machinery and robotics: A systematic review of architectures, quantified impacts, and deployment gaps," *ScienceDirect*, 2026. Available: <https://www.sciencedirect.com/science/article/pii/S2666154326004217>
15. Shailendra Mishra, Tariq Saleh M. Aldafas and Naif S. Alshammari, "A zero-trust digital twin framework for privacy-preserving multi-dataset intrusion detection in industrial IoT with lightweight blockchain auditing," *Scientific Reports*, 2026. Available: <https://www.nature.com/articles/s41598-026-42041-w>
16. Asif Ali Laghari, et al., "A novel and secure artificial intelligence enabled zero trust intrusion detection in industrial internet of things architecture," *Scientific Reports*, 2025. Available: <https://www.nature.com/articles/s41598-025-11738-9>
17. Carlos J. Ogayar-Anguita, et al., "Nested spatial data structures for optimal indexing of LiDAR data," *ISPRS Journal of Photogrammetry and Remote Sensing*, 2023. Available: <https://www.sciencedirect.com/science/article/abs/pii/S0924271622003112>
18. Xu Chen, et al., "Zero trust architecture for 6G security," *IEEE Network*, 2024. Available: <https://dl.acm.org/doi/10.1109/MNET.2023.3326356>
19. H. Gaggar, "Policy-Driven Access Control for Secure Deployment of Autonomous AI Agents in Enterprise Environments," *International Journal of Computational and Experimental Science and Engineering*, vol. 11, no. 4, 2025. Available: <https://doi.org/10.22399/ijcesen.5497>
20. D. Deshpande, K. C. Yaramati, C. T. Bobba, M. Priya, S. C. Pasupuleti, S. A. Gubbala, and N. Thakkar, "Development, Validity, and Test Reliability and Reproducibility of Facial Palsy-Associated Facial Movement: A Cross-Sectional Study," *Journal of Pharmacy & BioalliedSciences*, vol. 17, Suppl. 4, p. S3120, 2025.
21. A. Bhatnagar and V. R. K. Bonagiri, "Operational Excellence through Cloud-Native Distributed Systems," *International Journal of Computational and Experimental Science and Engineering*, vol. 12, no. 2, 2026. Available: <https://doi.org/10.22399/ijcesen.5207>