



OPEN
Compute Project



WHITE PAPER: TIME SYNCHRONIZATION IN AI DISTRIBUTED INFRASTRUCTURE

*Observability, Measurement, and Time-Triggered Coordination
for Distributed Training and Inference*

Open Compute Project, Time Appliances Project (TAP)

Version 0.9.1

Hesham Elbakoury, Independent Contributor

Ahmad Byagowi, Research Scientist, turbalance

White Paper | Draft for Community Review
October 1, 2026

Date: October 1, 2026



Executive Summary

Large-scale artificial intelligence systems distribute training and inference across accelerators, hosts, switches, and storage. This White Paper explains when a shared physical time base materially improves cross-host observability, one-way latency attribution, and explicitly time-triggered functions. It also draws an important boundary: synchronized wall clocks generally do not provide the correctness of collective communication, checkpoint consistency, causal tracing, or distributed transactions. Those properties normally come from protocol ordering, logical progress, versioned state, and trace relationships.

The paper compares NTP, IEEE 1588 PTP, software-based clock synchronization, logical clocks, hybrid logical clocks, and hardware timestamping. It organizes an illustrative architecture around three separate planes: time distribution, event timestamping, and temporal observability. Five figures, four comparative tables, and four use cases connect these mechanisms to straggler diagnosis, checkpoint operations, disaggregated inference, and causal tracing. Numerical scenario values are illustrative unless a cited source states otherwise, and the proposed architecture is not presented as a validated OCP Reference Architecture or Tested Configuration.

The intended audience includes OCP operators, timing and network engineers, AI infrastructure architects, accelerator and NIC implementers, observability teams, and standards participants. The contribution gives the Time Appliances Project a vendor-neutral vocabulary, evidence boundaries, design considerations, and open validation questions that can guide future interoperable profiles, reference architectures, and benchmarks.

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

Table of Contents

1. Introduction.....	4
2. Compliance with Open Compute Project Tenets.....	4
3. Fundamentals of Time Synchronization.....	5
4. Where AI Distributed Infrastructure Needs Precise Time.....	10
5. Time Synchronization in Distributed Training.....	12
6. Time Synchronization in Distributed Inference.....	15
7. Reference Architecture.....	17
8. Illustrative Use Cases.....	20
9. Comparative Analysis.....	21
10. Open Challenges and Future Directions.....	24
11. Design Considerations for AI Temporal Infrastructure.....	25
12. Conclusion.....	26
13. Glossary.....	28
14. References.....	29
15. License.....	33
16. About Open Compute Foundation.....	34

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

1. Introduction

The Open Compute Project Time Appliances Project provides a forum for open, interoperable time distribution and time-sensitive infrastructure. AI clusters make time especially relevant because events span many clock domains, including grandmasters, switches, NIC hardware clocks, host clocks, accelerator clocks, and software runtimes. This paper focuses on the operational and measurement value of relating those domains without treating synchronized physical time as a universal substitute for protocol correctness.

Distributed AI training and inference systems coordinate the work of thousands of GPUs or accelerators across many physical hosts, racks, and network fabrics. A recurring operational need in these systems is a sufficiently accurate and mutually comparable notion of “now” across hosts – not because most correctness properties depend on it directly, but because diagnosing performance problems, attributing latency, and reconstructing what happened across a distributed system all become far harder without it. When cross-host timestamps are not comparable, symptoms such as mis-attributed stragglers and untrustworthy observability traces become difficult to distinguish from genuine compute or network delay.

This paper surveys where and why time synchronization matters across the AI infrastructure stack and how it is implemented in practice, from GNSS-disciplined grandmaster clocks down to hardware timestamping in network interface cards (NICs). It combines background on established synchronization protocols with examples grounded in the operational realities of large-scale training clusters (e.g., ByteDance’s MegaScale system) and multi-tenant inference serving. Distributed-tracing and observability tooling for AI infrastructure – an area with active industry and research work, including efforts the author has contributed to – is used throughout as a motivating context for why comparable cross-host timestamps are operationally valuable, without claiming that any single such tool is authoritative.

The remainder of the paper is organized as follows. Section 3 reviews the fundamentals of clock synchronization technologies. Section 4 explains, at a systems level, where AI workloads are sensitive to time and where they are not. Sections 5 and 6 examine training and inference in turn, with worked examples. Section 7 presents a reference architecture. Section 8 collects illustrative use cases, labeled as hypothetical unless otherwise noted. Section 9 offers a comparative analysis in tabular form, framed as illustrative estimates rather than derived requirements. Section 10 discusses open problems, Section 11 distills the paper’s distinctions into practical design considerations, and Section 12 concludes.

2. Compliance with Open Compute Project Tenets

This contribution addresses the five OCP tenets as a vendor-neutral technical White Paper. The statements below describe design intent and community value; they are not measured deployment claims.

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

2.1. Openness

The paper builds on published protocols and specifications, including NTP, IEEE 1588 PTP, the OCP Data Center PTP Profile, and established logical-clock constructs. It separates standardized behavior, published research, vendor-reported results, and this paper's illustrative proposals so that operators and implementers can review or extend the work without depending on a proprietary product.

2.2. Efficiency

A fit-for-purpose timing objective avoids spending engineering effort on precision that a workload does not consume. Comparable timestamps can also shorten fault isolation by distinguishing compute, queueing, network, and clock-domain effects. The paper recommends measuring timestamp uncertainty and matching the synchronization method to the attribution-error budget rather than assuming that the tightest available clock is always necessary.

2.3. Impact

AI operators increasingly need to explain performance across thousands of accelerators and many independent clocks. A shared framework for time distribution, event timestamping, and temporal observability can improve the quality of incident analysis and performance attribution while reducing the risk of treating clock synchronization as a correctness mechanism where it is not one.

2.4. Scale

The analysis covers hierarchical timing distribution, multi-tier Clos fabrics, host and accelerator clock crossings, and cluster-wide observability. It highlights failure domains, holdover, profile selection, recalibration, and monitoring questions that become more important as participant count and clock-domain count increase.

2.5. Sustainability

Better temporal diagnosis can help operators identify idle accelerators, repeated work, and network or scheduling delays that waste compute capacity. Selecting only the accuracy needed by the workload can also avoid unnecessary timing complexity. Actual energy and resource effects remain implementation-specific and should be measured before a sustainability benefit is claimed.

3. Fundamentals of Time Synchronization

3.1. Network Time Protocol (NTP)

NTP is the long-standing Internet standard (RFC 5905; RFC 9769 subsequently defines interleaved modes that allow more accurate transmit timestamps) for synchronizing clocks over packet networks using a hierarchy of time sources (strata) rooted in reference clocks such as GNSS receivers or atomic clocks [3]. NTP is not inherently software-timestamped: implementations may timestamp in user space, the kernel or driver, or closer to the network interface, and achieved accuracy is deployment-dependent rather than an intrinsic protocol property. Commonly cited figures for best-effort software deployments on typical wide-area or shared local-area networks fall in the range of

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

roughly one to tens of milliseconds, but this range should not be treated as a guarantee: well-engineered deployments on a fast, lightly loaded LAN can do considerably better, adverse paths can do considerably worse, and a deployment claim should be validated by measuring offset and jitter for the actual servers, paths, and timestamp locations in use (Table 2) rather than assumed from this general figure. Even at its best-case end, this is adequate for log correlation, certificate validity, and coarse scheduling, but it is often insufficient for the tighter-coupling measurement needs found in AI training and inference, as later sections describe.

3.2. IEEE 1588 Precision Time Protocol (PTP)

The IEEE 1588 Precision Time Protocol synchronizes clocks in networked distributed systems using a hierarchy of PTP Instances. IEEE Std 1588g-2022 [2] defines the optional alternative terminology TimeTransmitter and TimeReceiver used in this paper for the older master and slave terms in IEEE Std 1588-2019 [1]. PTP does not require hardware timestamping and does not guarantee a fixed accuracy. Achieved maximum time error depends on the selected profile, timestamp location (PHY, MAC, driver, or application), oscillator quality, topology and hop count, path-asymmetry calibration, message intervals, and servo design. A deployment claim should therefore identify the profile and topology and report validated maximum time error under stated conditions rather than assign a universal PTP accuracy range. Specialized high-accuracy profiles have demonstrated nanosecond-scale results in calibrated networks (Section 7.1), but those results are profile- and implementation-specific. PTP's delay request-response exchange (Figure 1) allows a TimeReceiver to compute offset and network path delay from four timestamps (t_1 – t_4). The usual computation assumes symmetric forward and reverse path delay; unknown asymmetry appears directly as clock-offset error, and tighter profiles apply an explicit delayAsymmetry correction rather than relying on the basic exchange to infer it.

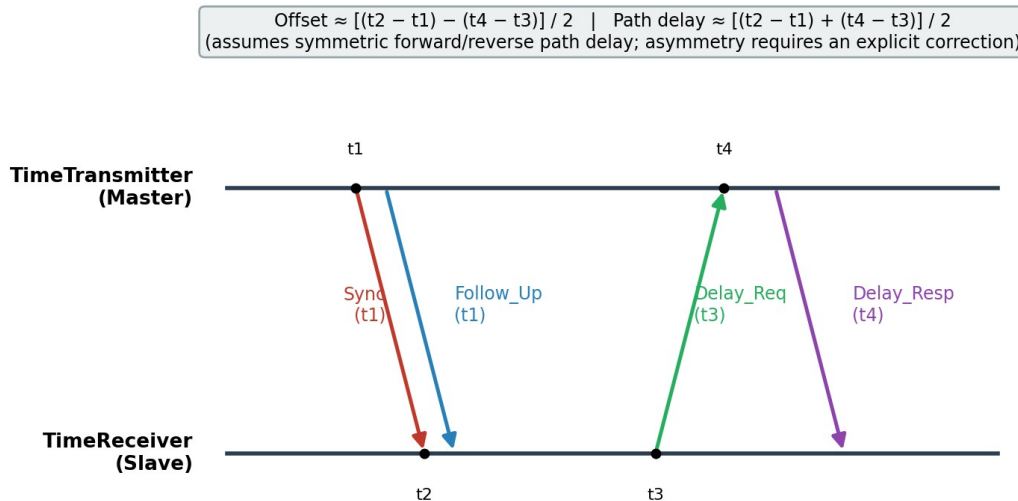


Figure 1. IEEE 1588 PTP delay request-response exchange: the TimeReceiver computes offset and path delay from four exchanged timestamps, under a symmetric-delay assumption.

Boundary and transparent clocks – distinct mechanisms detailed in Section 7.1 – are the two standard ways to limit accuracy degradation as PTP messages traverse multiple hops in the multi-tier Clos or fat-tree fabrics common in AI data centers [6, 7]; a boundary clock acts as a TimeReceiver on its upstream port and a TimeTransmitter on downstream ports, while a transparent clock forwards PTP event messages and accumulates measured residence time in the correctionField. Path asymmetry, timestamp error, oscillator noise, and servo behavior remain with either mechanism, and the achieved improvement depends on the selected PTP profile. Hardware timestamping implemented in the NIC or switch ASIC, where used, avoids the jitter introduced by operating-system scheduling, interrupt coalescing, and network stack traversal that commonly limits software-timestamped deployments.

Redundant grandmasters, Best Master Clock Algorithm (BMCA) or profile-specific selection, and oscillator holdover are the normal IEEE 1588 mechanisms for tolerating a grandmaster failure. Separately, research has proposed collaborative slave-clock correction, in which surviving TimeReceivers cross-estimate a reference clock among themselves to sustain sub-microsecond accuracy for a period after a grandmaster outage; this is a specific experimental enhancement rather than standard production practice, and it should be evaluated as an addition to – not a replacement for – conventional BMCA-based failover [5].

3.3. Software-Based Nanosecond Synchronization: HUYGENS

An important alternative to hardware-timestamped PTP is software synchronization implemented through a software algorithm using commodity NIC timestamp mechanisms. HUYGENS, developed

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

at Stanford and Google, demonstrated that a probe-based algorithm running on commodity hosts, using coded probes to reject queuing-delay and jitter noise, support-vector-machine estimation of one-way propagation time, and a “natural network effect” in which pairwise-synchronized clocks are cross-checked for transitive consistency, can synchronize clocks to within a few tens of nanoseconds under controlled, varying-load testbed conditions using standard NIC timestamp mechanisms rather than specialized timing hardware [8]. The published evaluation covers specific hardware testbeds rather than a documented, large-scale production deployment, so the result should be read as evidence of what the technique can achieve under those conditions rather than a general production guarantee. Applying it to arbitrary GPUs, switches, cloud environments, or multi-tenant fabrics would require defining the necessary timestamp support and validating it under those conditions. A companion system, SIMON, extends the same probe infrastructure to reconstruct fine-grained network state from end-host measurements [9], and On-Ramp uses the resulting one-way-delay accuracy to detect and react to transient congestion in datacenter packet transport [10]. The practical significance is that a software synchronization algorithm can achieve near-PTP-grade results when the evaluated NIC timestamp mechanisms and testbed conditions are present. Industry AI-fabric observability platforms report building on this pattern [18, 19, 20], but those vendor claims are not independently validated evidence.

3.4. Logical and Hybrid Logical Clocks

Physical-time synchronization is necessarily bounded by residual uncertainty. Lamport’s logical clocks established that a consistent causal (happened-before) ordering of events can be maintained without physical time at all, using monotonically increasing counters exchanged with messages [13]. Vector clocks extend this to precisely capture concurrency but scale poorly with node count. Hybrid Logical Clocks (HLC) combine a physical-time component with a logical counter; the causal ordering HLC provides is governed by its logical update rules and remains valid even when physical synchronization degrades, while only HLC’s closeness to wall-clock time inherits the underlying physical-clock bound [11]. Google’s TrueTime API takes a related but distinct approach: in Google’s original Spanner deployment, it exposes an explicit, deployment-defined time-uncertainty interval (backed by GPS and atomic clock ensembles in that specific implementation) rather than a point estimate, so that distributed transactions can reason correctly about ordering despite bounded clock skew [12]. These software-level constructs are complementary to, not substitutes for, hardware time synchronization: the tighter the physical synchronization bound, the smaller the uncertainty interval HLC- or TrueTime-style systems must tolerate.

3.5. Hardware Timestamping in AI Accelerator Fabrics

SmartNICs and RDMA-capable NICs that interconnect AI accelerators increasingly expose a PTP hardware clock (PHC); for example, official NVIDIA documentation describes a Linux PHC exposed on ConnectX NICs, accessible through the standard Linux PTP hardware-clock character-device interface (`/dev/ptp*`) and the `PTP_SYS_OFFSET` family of ioctls, which also support cross-timestamping a PHC against the host’s `CLOCK_REALTIME`. GPUs are a separate case: they generally expose independent device-domain timestamps through mechanisms such as CUDA/CUPTI rather than an IEEE 1588 PHC of their own, and correlating those GPU-domain

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

timestamps with NIC PHC time requires explicit cross-domain calibration rather than being automatic.

At a sketch level, that calibration chain has two distinct links plus an ongoing re-calibration need, rather than a single step: (1) NVIDIA's own CUPTI documentation states that GPU activity timestamps are recorded on the GPU in its own clock domain and are converted to a CPU timestamp domain by linear interpolation during post-processing – this is a documented, driver-level conversion, not a timestamp captured live at submission or completion, and its accuracy depends on the specific CUPTI activity record and any latency-timestamp option enabled, which an implementation would need to identify explicitly [48]; (2) separately, the host CPU clock is kept close to the NIC's PHC using host-to-PHC cross-timestamping (e.g., via the `PTP_SYS_OFFSET_PRECISE` ioctl or equivalent kernel support), which pairs a `CLOCK_REALTIME` reading with the corresponding PHC reading captured at the same instant; and (3) periodic re-calibration is needed because each link's offset drifts independently – the GPU-to-CPU interpolation is subject to the GPU/CPU clock drift between calibration points, and host-to-PHC correlation is subject to the residual PTP synchronization error discussed in Section 3.2. This is offered as a sketch of the mechanism, not a full specification: production systems differ in how frequently they re-calibrate each link and how they bound the resulting compound error across both links, and that implementation-level detail remains an open area this paper does not fully resolve (Section 10).

Aligning an accelerator's event timeline (kernel launch, collective-communication start/end) with a PTP-synchronized NIC's PHC allows fine-grained, cross-host correlation of compute and network events, which supports both straggler diagnosis in training and causal tracing in inference, discussed in Sections 5 and 6; this is primarily an observability and measurement capability rather than a mechanism that itself makes training or inference correct. In production practice, commercial software-driven AI fabric platforms report combining a fleet-wide software clock synchronization layer with continuous probe-based telemetry – an architecture the vendor describes as descended from HUYGENS/SIMON – to correlate RDMA queue-pair, NCCL collective, and switch-level events across GPU fleets for straggler and fault localization [18, 19, 20]. These are vendor claims (the vendor's own material varies between describing “sub-microsecond fleet alignment” and “nanosecond telemetry”); they are reported here as an illustration of an operational pattern, not as independently validated accuracy figures.

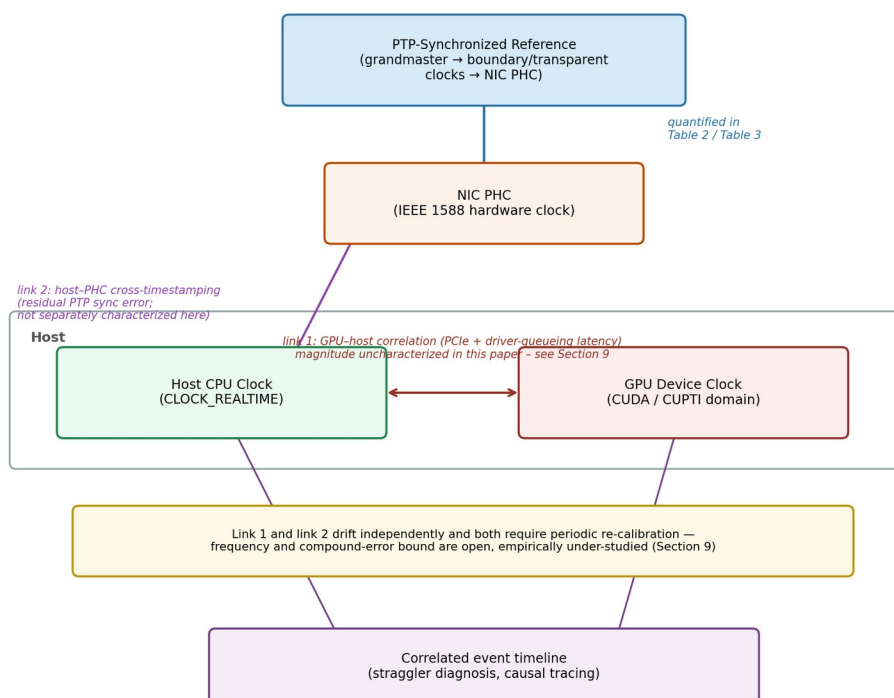


Figure 2. GPU, CPU, and NIC clock domains and the calibration links connecting them. Link 1 (GPU–host) and link 2 (host–PHC) are each subject to their own drift and are not separately characterized in this paper; only the PTP-synchronized reference above the NIC PHC is quantified in Tables 2–3.

4. Where AI Distributed Infrastructure Needs Precise Time

AI distributed systems differ from many classical distributed workloads in that they involve frequent, tightly-coupled coordination across large numbers of participants [22]. It is worth separating two distinct roles time synchronization can play, because the two are often conflated: (a) as an observability and measurement substrate that makes cross-host behavior legible – comparable timestamps, one-way latency attribution, and temporal correlation for debugging and diagnosis; and (b) as an operational input to functions that are explicitly defined in terms of physical time, such as a scheduled wall-clock trigger or a deadline. Role (a) applies broadly across the examples in this paper. Role (b) applies more narrowly, to specific mechanisms that are deliberately designed around physical time. Correctness properties such as collective-communication ordering, checkpoint consistency, and causal tracing are, in most designs, governed by protocol ordering, logical progress, versioned state, and trace relationships rather than by synchronized wall clocks – clock offset does not itself stall a collective or make a checkpoint inconsistent. Four properties nonetheless make AI infrastructure a setting where comparable time is unusually valuable:

- Bulk-synchronous execution: data-parallel and pipeline-parallel training advance in steps gated by collective operations (AllReduce, AllGather, pipeline bubbles). NCCL-style

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

collective correctness depends on matching calls, ranks, streams, and consistent operation order rather than synchronized wall time, but comparable cross-host timestamps are what let engineers see which participant is delayed and why [12, 22, 23].

- Tight latency budgets in inference: interactive and agentic inference services operate against millisecond-scale service-level objectives (SLOs) for time-to-first-token and inter-token latency, where clock skew across disaggregated prefill/decode pools can corrupt latency accounting even when the underlying request handling is correct.
- Causal complexity: a single training step or inference request touches dozens of hosts, network hops, and asynchronous queues. Causal reconstruction for debugging does not strictly require synchronized timestamps – trace/span identifiers and parent-child message relationships can establish causal order on their own – but comparable timestamps add temporal placement and latency attribution that identifiers alone do not provide (Section 6.2).
- Scale amplifies small errors: with thousands of participants, even a small per-node probability of a measurement or diagnosability gap becomes a near-certainty somewhere in the cluster on every training step or every second of inference traffic, making time-correlated observability disproportionately valuable at scale.

These properties motivate two complementary applications of time synchronization in AI infrastructure: as an observability substrate that makes system behavior legible to humans and automated tooling (the dominant use case in this paper, e.g., hardware-timestamped distributed tracing), and, more narrowly, as an operational input for functions explicitly defined around bounded physical time or scheduled deadlines (e.g., a coordinated checkpoint-trigger schedule, discussed with its consistency caveats in Section 5.2).

Table 1 makes this distinction concrete by classifying, function by function, what actually provides correctness and what role (if any) time synchronization plays. The pattern that recurs throughout the rest of the paper is visible here at a glance: synchronization is rarely the correctness mechanism itself, but it is very often the difference between an attributable and an unattributable failure.

Table 1. Classification of AI infrastructure functions by what provides their correctness versus the role time synchronization plays.

AI Infrastructure Function	What Actually Provides Correctness	Role of Time Synchronization
Collective communication (NCCL AllReduce/AllGather)	Matching calls, ranks, streams, and consistent operation order (Sections 4.1, 4.4)	Observability: attributes stalls/delay to a specific host or link
Checkpoint consistency	Coordinated step/version identifier, quiescence or snapshot semantics, atomic metadata commit (Section 4.2)	Scheduling input: bounds trigger-time skew across shards
Causal / distributed tracing	Trace/span identifiers and parent-child message relationships (Section 5.2)	Observability: adds temporal placement and latency attribution on top of causal order
Inference SLA / TTFT measurement	Per-stage instrumentation and event records (Section 5.1)	Observability: enables cross-host latency attribution
Rate limiting / quota enforcement	Monotonic local clock plus authoritative shared state (Section 5.4)	Not required for correctness

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

AI Infrastructure Function	What Actually Provides Correctness	Role of Time Synchronization
Checkpoint or maintenance scheduling	Explicit deadline/schedule definition in the application	Scheduling input: the function is defined in terms of physical time
General log correlation / audit	Timestamped records across independent logs	Observability aid for manual reconciliation

5. Time Synchronization in Distributed Training

5.1. Synchronous Collective Communication and Stragglers

Large language model (LLM) pretraining jobs combine data, tensor, and pipeline parallelism across thousands of GPUs, and every strategy requires frequent cross-device coordination to maintain model consistency [22]. A microbatch delayed anywhere in a pipeline-parallel stage propagates delay into the surrounding data-parallel group, and stragglers – slow workers – can arise from causes far more varied than outright hardware failure, including transient thermal throttling, NUMA or PCIe contention, or asymmetric network congestion [22]. It is worth separating what is diagnosable locally from what requires cross-host comparison: message identifiers, CUDA event timestamps, operation ordering, and local NIC telemetry can already reveal many sub-millisecond causes on a single host without any globally synchronized clock at all. What accurate, comparable synchronization adds is specifically cross-host timestamp alignment and one-way latency attribution, which materially narrow the search only once a cause is suspected to span multiple hosts. Without that cross-host alignment, engineers are often left correlating coarse, independently timestamped logs that cannot cleanly distinguish a slow compute kernel on one host from a delayed network send on another (Figure 3).

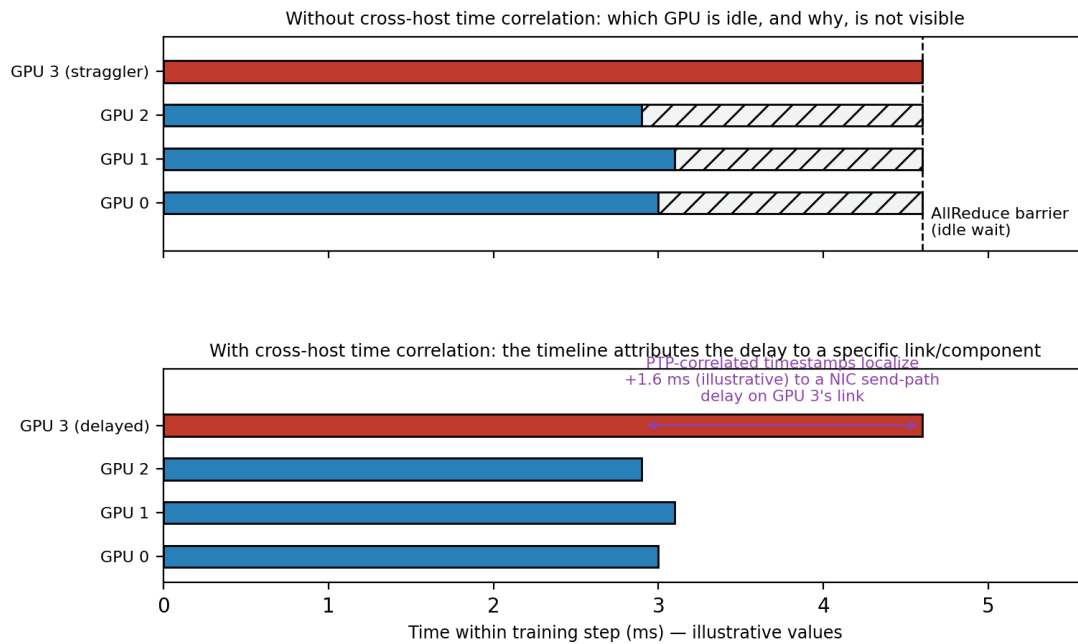


Figure 3. Illustrative example of cross-host time correlation aiding straggler diagnosis in synchronous data-parallel training; the +1.6 ms value and root cause shown are hypothetical, not measured.

Production systems such as MegaScale report using fine-grained CUDA event recording combined with system-wide, time-correlated heat-maps and 3-D parallelism visualizations to troubleshoot straggler-induced slowdowns at scale; the published account describes CUDA-event timing rather than a PTP-based synchronization layer specifically [21]. A five-month trace-driven study of a large production LLM training cluster found that stragglers are common, frequently transient, and attributable to a range of interacting causes – underscoring that time-correlated, fine-grained tracing, rather than coarse aggregate metrics, is valuable for diagnosing them [22]. Commercial platforms report a similar operational pattern, using fleet-wide software clock synchronization to help localize straggler-inducing link, NIC, or GPU anomalies to a specific component; this is a vendor-reported claim rather than independently validated evidence [18, 19].

5.2. Checkpoint Consistency

Periodic checkpointing of model, optimizer, and data-loader state must capture a globally consistent snapshot across all data-parallel replicas and pipeline/tensor-parallel shards; an inconsistent snapshot (for example, one shard reflecting step N and another step N–1) can silently corrupt recovery. Distributed filesystems used for large-scale checkpointing, such as Meta’s Tectonic or ByteDance’s use of HDFS, are engineered to let thousands of GPUs save and load checkpoints simultaneously while preserving this consistency [24]. Starting writes at nearly the same wall-clock instant does not, by itself, guarantee that every shard represents the same logical training step: consistency requires a coordinated checkpoint epoch or step/version identifier, quiescence or copy-

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

on-write snapshot semantics, and atomic commit of checkpoint metadata – the checkpoint protocol, not clock alignment, is what provides correctness. A synchronized wall-clock deadline, disseminated via PTP-synchronized hosts, is nonetheless a useful way to schedule and diagnose when a coordinated checkpoint should begin, and to bound how far shard write-start times can drift apart, without requiring an expensive full-cluster stop-the-world barrier on every step.

5.3. Asynchronous and Semi-Synchronous Training

Not all training is strictly bulk-synchronous. Local-SGD and staleness-bounded approaches (e.g., ASHL, EDiT, and Google DeepMind's DiLoCo and Decoupled DiLoCo) relax per-step synchronization to tolerate heterogeneous worker speeds and even geographically distributed, poorly connected worker pools [23, 25, 30, 44]. These methods normally express staleness in algorithmic terms – steps, versions, tokens, updates, or synchronizer rounds – rather than physical age; Decoupled DiLoCo, for example, bounds staleness through a quorum, a grace window, and token-weighted aggregation rather than a wall-clock deadline. Physical clock synchronization becomes relevant to these methods specifically when an implementation explicitly defines a wall-time age or deadline; where staleness is tracked algorithmically, a shared logical notion of “how far behind” a worker is degrades gracefully on its own terms and does not depend on physical-clock accuracy.

5.4. Collective Communication Libraries and Large-Scale Training Systems

The AllReduce, AllGather, and reduce-scatter primitives referenced throughout this section are implemented in collective-communication libraries such as NVIDIA's NCCL [31] and the broader family of algorithms surveyed by Thakur, Rabenseifner, and Gropp for MPICH [37] and by Renggli et al. for deep learning generally [39]; industry surveys of vendor-led collective libraries document how each vendor's implementation differs in mechanism [45], and empirical performance evaluations of collective operations across frameworks further quantify these differences [46]. As established in Table 1, NCCL and comparable libraries coordinate correctness through matching collective calls, ranks, streams, and consistent operation order, not synchronized wall time. Systems research has repeatedly shown that network behavior, not raw compute, is frequently the binding constraint on training throughput at scale [38], motivating techniques such as multi-rail collective communication that spread traffic across several physical NICs per host, and generic high-performance collectives such as Blink that adaptively route around topology irregularities [47]. These considerations are amplified by the sheer scale of frontier training systems: large language model pretraining runs such as GPT-4 [40] and Megatron-LM-based systems training hundreds of billions to trillions of parameters across thousands of GPUs [42, 43] depend on memory-optimization frameworks like ZeRO-Infinity [41]; where time synchronization helps at this scale, it is again as an observability and cross-host measurement aid – for example, in diagnosing why a collective is slow. Industry and research accounts of operating frontier clusters describe data-movement and network-latency limits as an increasingly significant constraint on further scaling [36]; that source's evidence does not extend to power or general coordination-overhead constraints, which this paper does not otherwise document with a citation.

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

6. Time Synchronization in Distributed Inference

6.1. Disaggregated Prefill/Decode Pipelines

Modern high-throughput LLM inference frequently disaggregates the compute-bound prefill phase from the memory-bandwidth-bound decode phase across separate GPU pools, with KV-cache state transferred between them. Measuring and enforcing per-stage latency SLOs (time-to-first-token, inter-token latency) requires that timestamps recorded at the gateway, prefill pool, KV-cache transfer layer, and decode pool be mutually comparable; unaccounted skew between hosts is otherwise indistinguishable from genuine latency, leading to incorrect SLA attribution and misdirected optimization effort (Figure 4).

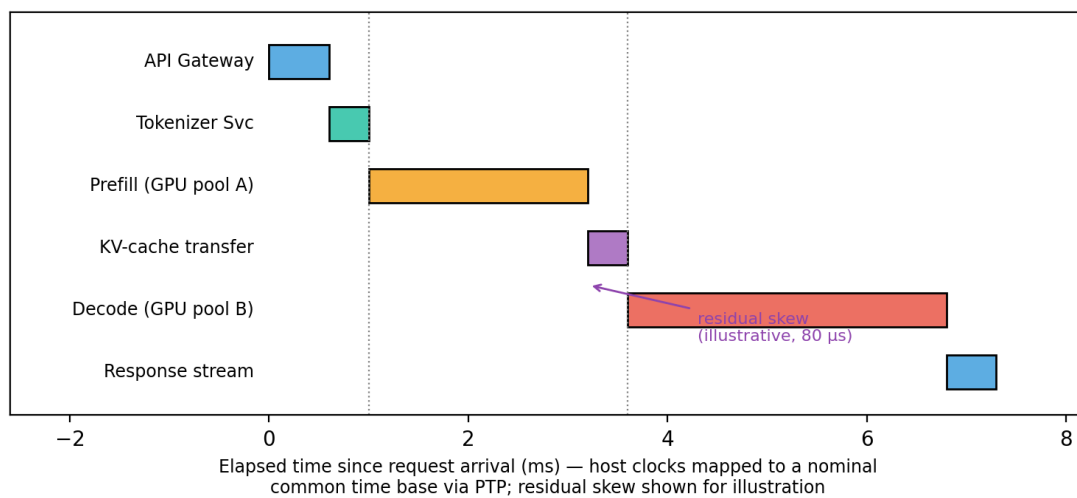


Figure 4. An end-to-end inference request trace spanning disaggregated prefill and decode pools; host clocks are mapped to a nominal common time base via PTP, with residual skew (here, an illustrative 80 μs) shown for illustration rather than measured across a real deployment.

6.2. Multi-Tenant Tracing and Causal Correctness

In multi-tenant inference platforms serving many concurrent requests across shared accelerator pools, causal reconstruction of a request's path does not strictly require synchronized timestamps: trace/span identifiers, parent-child relationships, and message edges are what establish causality, and this is essential for debugging, capacity planning, and auditability regardless of clock quality. Comparable timestamps add temporal placement and latency attribution on top of that causal structure, and they matter because naive timestamp-based causal reasoning can fail silently when skew exceeds safe bounds. The most directly relevant empirical evidence available reports no observed causal-ordering violations through roughly 3 ms of clock skew in its tested setup, with violations appearing by roughly 5 ms, while conventional latency and throughput SLOs remained largely unaffected in that range [15]; this single tested setup does not establish a general threshold – the skew at which violations appear should depend on the specific workload's inter-event timing, the inference pipeline's topology and hop count, and the causal-reconstruction algorithm used, none of

which this paper varies or evaluates. It motivates keeping clock skew well below the observed range where causal reconstruction matters and treating the specific numbers as one data point rather than a validated bound, and it does not establish a sub-microsecond synchronization requirement for inference generally. Hybrid approaches that combine physical time with explicit uncertainty bounds or logical causality tracking are one way to make this reasoning more robust [11, 15, 16], and this parallels long-standing distributed-tracing practice, in which systems such as Dapper capture low-overhead, causally linked traces of requests as they propagate through large distributed services primarily via identifiers and parent-child links rather than synchronized clocks [14].

6.3. Agentic and Tool-Using Inference Workloads

Agentic AI systems, where a model orchestrates multiple tool calls, sub-model invocations, and asynchronous external services within a single logical request, compound the causal-tracing challenge: a single user-facing request may fan out into a tree of causally related but temporally interleaved operations across heterogeneous services with independently synchronized clocks. Reliable ordering here benefits from the same identifier-based causal tracking and hybrid-clock techniques used in training-cluster observability, applied at the granularity of individual tool calls and sub-agent invocations rather than GPU kernels; comparable timestamps remain useful for temporal placement and latency attribution even though causality itself is carried by trace identifiers and message edges.

As a concrete, hypothetical illustration: a planning sub-agent issues two tool calls – a retrieval call and a code-execution call – that run concurrently and return to a synthesis step at different times. Each event record in such a trace would plausibly carry three things: a physical timestamp for temporal placement (“when approximately did this happen”), a trace/span identifier with a parent-child link for causal order (“what caused what,” independent of clock quality), and, where the implementation supports it, a Hybrid Logical Clock value for causal consistency that degrades gracefully if the physical timestamps across the independently synchronized services involved are not perfectly comparable. None of the three fields is redundant: dropping the physical timestamp would make latency attribution across tools impossible; dropping the trace identifiers would make causal order dependent on clock accuracy in exactly the way Section 6.2 shows can fail silently under skew.

6.4. SLA Enforcement and Rate Limiting

Token-bucket rate limiting and per-node latency deadlines primarily need a reliable, monotonic elapsed-time source at each node rather than a globally synchronized wall clock; a monotonic local clock is generally sufficient to measure elapsed time correctly even without cross-host synchronization. Distributed, cluster-wide quota correctness is a related but separate problem that additionally requires authoritative shared state or a replica-consistency design, independent of clock synchronization. SLA time-to-first-token measurement and billing/accounting reconciliation are further separate concerns: SLA measurement benefits from comparable cross-host timestamps in the sense described in Section 6.1, while billing correctness depends on authoritative event records rather than clock agreement alone. Coarse or skewed clocks can still cause a serving node's

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

reported latency to be misattributed relative to the measurement point, which is the main way clock quality affects this area.

7. Reference Architecture

The architecture in Figure 5 is organized around three functionally distinct planes, which this paper names explicitly here because they are easy to conflate in practice: a time-distribution plane (GNSS through the PTP grandmaster/boundary-clock hierarchy to each host's PHC) that establishes a common time base; an event-timestamping plane (the point at which a GPU, NIC, or switch actually records a timestamp for a specific event, in hardware or software) that determines how accurately an event's occurrence is captured relative to that time base; and a temporal-observability plane (the collection, correlation, and presentation layer, e.g., distributed tracing) that consumes timestamped events but does not itself distribute time. Sections 3 and 3.5 correspond to the first two planes respectively, and Sections 5–6 draw on the third; keeping the three separate matters because a fault or limitation in one (e.g., a delayed PTP sync) does not necessarily imply a fault in another (e.g., correct event ordering, which the observability plane can still establish via trace identifiers per Section 6.2).

Figure 5 illustrates a representative time-synchronization hierarchy for an AI training/inference cluster; the PTP clock-role mapping shown here is this paper's own proposed architecture rather than a deployed reference design, and any real deployment should validate it against a specific PTP profile. Two fault-separated external primary references (e.g., independent GNSS receivers, each with its own source-health, spoofing/jamming detection, and holdover monitoring) each feed a redundant PTP grandmaster, so that grandmaster redundancy also provides independent time-source redundancy rather than two grandmasters sharing a single point of failure upstream; each grandmaster is characterized by profile-specific clockClass and timeSource attributes rather than by an NTP stratum number – NTP stratum terminology and PTP grandmaster attributes describe related but distinct hierarchies and should not be conflated. ToR and spine switches act as boundary clocks in this design (Section 7.1 discusses why boundary and transparent clocks are distinct roles rather than interchangeable labels); each GPU host's SmartNIC maintains a PTP hardware clock (PHC) synchronized to the grandmaster, and this PHC – not the observability plane – is what distributes time to the training and inference runtimes (collective communications, checkpointing, straggler diagnosis; request tracing, KV-cache transfer, SLA scheduling). The observability plane at the base of the figure is a separate telemetry-collection path: it consumes timestamped events from the runtimes and hosts to correlate them, but it does not itself serve as a clock source for either runtime, as the figure's solid (time-distribution) versus dashed (telemetry) lines indicate. Grandmaster failover follows standard BMCA/profile-specific selection plus oscillator holdover bounded by a stated holdover limit, shown separately from the main hierarchy; the collaborative slave-clock correction discussed in Section 3.2 is a further, experimental enhancement rather than part of this baseline design.

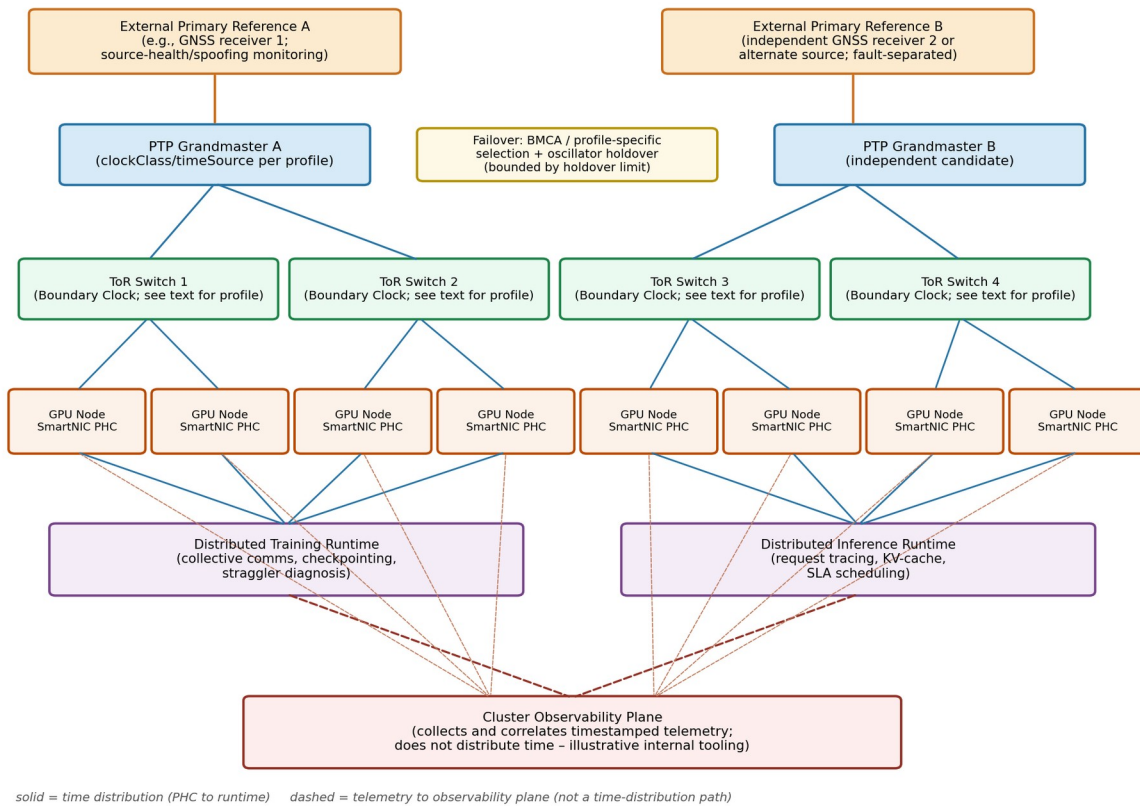


Figure 5. Reference time-synchronization hierarchy, organized around the time-distribution, event-timestamping, and temporal-observability planes described above – this paper's own proposed architecture – spanning two fault-separated external primary references, redundant PTP grandmasters with BMCA-based failover and a stated holdover limit, boundary-clock switches, per-host hardware clocks that distribute time to the runtimes (solid lines, time-distribution plane), and a separate observability plane that only consumes telemetry (dashed lines, temporal-observability plane).

This layered design reflects established practice: independent external primary references anchor the top of the hierarchy; hardware timestamping at the NIC or switch, where used, avoids operating-system jitter; and redundant, fault-separated grandmasters bound the accuracy degradation that follows a reference or master failure [1, 5, 6]. The observability plane at the base of the figure is where hardware timestamps become actionable: it is the layer responsible for correlating GPU, NIC, and switch events into coherent, time-ordered traces – the approach taken, for example, in hardware-timestamped distributed tracing designs for large-scale AI infrastructure – without itself being a source of time for the runtimes it observes.

7.1. Interconnect and Switch Fabrics

Boundary clocks and transparent clocks are distinct PTP behaviors, not interchangeable labels. A boundary clock terminates and regenerates PTP at each hop, acting as a TimeReceiver upstream

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

and a TimeTransmitter downstream. A transparent clock forwards PTP event messages while measuring residence time and accumulating the required correction in the correctionField; end-to-end and peer-to-peer transparent clocks account for path information differently and must not be conflated. Figure 5 illustrates a boundary-clock design at the ToR tier. A transparent-clock design would use the same physical switches differently and would need its own profile, transport, delay mechanism, message intervals, domain, and maximum-time-error budget. The Open Compute Project Data Center PTP Profile v1.1, published in November 2022 [52], is a concrete data-center profile built around a transparent-clock chain, IPv6 transport, the end-to-end delay request-response mechanism, and explicit network limits and error budgets. A deployment should select and validate one coherent profile instead of mixing that transparent-clock model with the boundary-clock illustration without an explicit design decision.

The physical fabrics that carry data traffic in Figure 5 are themselves heterogeneous – relevant here mainly because it implies heterogeneous clock domains, not as a survey of AI interconnects in their own right. Scale-up interconnects such as NVIDIA’s NVLink/NVSwitch [26], AMD’s Instinct MI300X interconnect [35], and the vendor-neutral UALink specification [27] (whose current catalog also includes separate Common 2.0 and 200G Data Link and Physical Layers 2.0 documents released in 2026, beyond the original 1.0 release) link GPUs within a node or rack. Scale-out fabrics such as the Ultra Ethernet Consortium’s Ultra Ethernet Specification [29] and the Open Compute Project’s ESUN Base Specification Rev. 1.0, released February 9, 2026 [28], target inter-node AI/HPC traffic. None of these specifications themselves define an IEEE 1588 PTP mapping; where PTP support exists on a given device, it coexists independently and should be cited to the specific vendor claim – e.g., Broadcom’s Tomahawk 5 documents PTP and SyncE support [34] – rather than assumed as full protocol integration. For fabrics that need accuracy beyond commodity PTP, the IEEE 1588-2019 High Accuracy Profile, exemplified by White Rabbit’s sub-nanosecond synchronization [32], is the relevant reference point.

7.2. Datacenter Topology and Centralized Fabric Control

The Clos-based, centrally controlled datacenter network topology popularized by hyperscale operators provides the multi-tier switch hierarchy (spine, aggregation, ToR) that this paper’s proposed PTP clock-role mapping is layered onto in Figure 5 [33]; the cited source establishes the Clos topology and centralized control design, not the PTP clock-role assignment itself, which is this paper’s own architectural proposal and should be validated against a specific PTP profile before deployment. Centralized software control of the fabric, in the same spirit as centralized routing control, also creates an opportunity to centrally manage and monitor the PTP grandmaster/boundary-clock hierarchy rather than relying purely on distributed protocol convergence; a data-movement and network-latency study of frontier AI training infrastructure describes networking overhead as an increasingly significant constraint at large cluster scale [36], though that source’s evidence is specific to data-movement and network latency and should not be read as establishing power or general coordination-overhead constraints as well.

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

8. Illustrative Use Cases

The four scenarios below are hypothetical, constructed to illustrate the mechanisms discussed in Sections 5–7; specific numeric values (percentages, millisecond figures, GPU counts) are illustrative rather than measured production data unless a citation is given. They are intended to show how time synchronization would plausibly be used, not to report field measurements.

8.1. Use Case 1: Diagnosing an Intermittent Straggler in a 4,096-GPU Pretraining Job

In this hypothetical scenario, a data-parallel training job exhibits a periodic step-time regression roughly every 40 steps, but per-GPU utilization metrics (sampled at 1-second granularity) show nothing unusual. With PTP-synchronized hardware timestamps correlating GPU kernel completion events with NIC send/receive timestamps, engineers could align the AllReduce start times of all participants on a single time axis and localize a recurring delay to one rack's NIC send path, coincident with a scheduled thermal-throttling event – illustrating how cross-host time correlation can make a root cause visible that per-node aggregate metrics alone would miss (Figure 3). The clock correlation is the measurement method here; the underlying fault is a real transmission-path delay, not the clock itself.

8.2. Use Case 2: Consistent Multi-Shard Checkpointing

A 512-GPU job using 3-D parallelism (data, tensor, pipeline) must checkpoint every 30 minutes without stalling training for more than a few seconds. A synchronized wall-clock deadline, disseminated via PTP-synchronized hosts, can schedule all shards to begin the checkpoint write at nearly the same instant, bounding trigger skew across the cluster. Consistency of the resulting checkpoint, however, still depends on a coordinated checkpoint epoch or step/version identifier, quiescence or copy-on-write snapshot semantics, and atomic commit of checkpoint metadata; PTP synchronization aligns the scheduled trigger, but the checkpoint protocol – not clock alignment – is what makes the checkpoint globally consistent.

8.3. Use Case 3: SLA-Compliant Multi-Tenant Inference Serving

A shared inference platform serving multiple enterprise tenants must enforce a 200 ms time-to-first-token SLO per request while disaggregating prefill and decode across separate GPU pools connected by RDMA. PTP-synchronized hardware timestamps at the gateway, prefill, KV-cache transfer, and decode stages would let the platform attribute latency to each stage (Figure 4), distinguishing genuine compute-bound delay from residual clock skew, and supporting SLA-credit calculation and capacity planning.

8.4. Use Case 4: Cross-Service Causal Tracing for an Agentic Workflow

An agentic assistant fans a single user request out into a document-retrieval call, two independent tool invocations, and a final synthesis call across four independently operated services. Trace identifiers and parent-child relationships – for example, W3C Trace Context's trace ID, parent ID, and flags, which standardize identifier propagation but do not themselves define Hybrid Logical Clock propagation – establish the causal ordering of sub-operations for debugging and cost attribution. An HLC value, if used alongside this, would need to be carried as a custom vendor-

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

specific field or a separate context item with explicitly defined send/receive update rules, complementing rather than replacing the trace-identifier structure, even though the underlying physical clocks of the four services are synchronized to different, imperfectly aligned NTP sources [11, 16].

9. Comparative Analysis

The tables below summarize, at a high level, how the synchronization approaches discussed in Section 3 differ in mechanism and typical role. Physical-time and logical-time approaches are compared separately (Sections 9.1 and 9.2) because they answer different questions – what wall-clock performance a physical-time technology can be validated to achieve versus what ordering or consistency guarantee a logical-time construct provides – and putting them in one table under a single “accuracy” column understates that distinction.

9.1. Physical-Time Synchronization Technologies

Table 2. Comparison of physical-time synchronization technologies by timing mechanism, evidence basis, and primary AI-infrastructure role.

Technology	Evidence-Bounded Performance Statement	Timestamping Point	Primary AI Infrastructure Role
NTP	Deployment-specific; measure offset and jitter for the selected servers, paths, timestamp locations, and operating conditions [3]	Implementation-dependent (user space, kernel, or driver)	Log correlation, coarse scheduling, non-critical control plane
IEEE 1588 PTP	Profile- and topology-specific; validate maximum time error for the deployed clock chain [1, 52]	Implementation-dependent; commonly hardware (NIC/switch) for tighter accuracy	Collective-comm correlation, straggler diagnosis, cluster tracing
IEEE 1588-2019 High Accuracy Profile (e.g., White Rabbit-derived)	Sub-nanosecond demonstrated in calibrated White Rabbit networks [32]; not a general PTP guarantee	Hardware + delay-asymmetry compensation	High-precision fabrics, fine-grained kernel-level tracing

PTP with hardware timestamping is the relevant default once sub-millisecond cross-host attribution matters, as in straggler diagnosis or distributed tracing (Sections 5.1, 5.2); NTP is normally sufficient for coarse scheduling and log correlation, where millisecond-scale accuracy is tolerable. The IEEE 1588-2019 High Accuracy Profile is worth its added engineering cost mainly for specialized fabrics that need nanosecond-scale alignment, not as a general-purpose upgrade.

9.2. Logical-Time and Uncertainty-Interval Constructs

Table 3. Logical-time and uncertainty-interval constructs relevant to AI distributed infrastructure, compared by the ordering or consistency guarantee each provides rather than by wall-clock accuracy [11, 12, 13].

Construct	What It Guarantees (vs. What It Says About Wall-Clock Accuracy)	Primary AI Infrastructure Role
Logical clocks (Lamport)	Causal (happened-before) ordering only; carries no wall-clock information and makes no accuracy claim	Causal ordering without physical-time dependency

Construct	What It Guarantees (vs. What It Says About Wall-Clock Accuracy)	Primary AI Infrastructure Role
Hybrid Logical Clocks	Causal ordering that remains valid regardless of physical-clock accuracy; its closeness to wall-clock time is bounded by whatever physical-time source (e.g., NTP or PTP) it is paired with	Consistent snapshots and causal tracing across imperfectly synced hosts
TrueTime-style Intervals	An explicit, deployment-defined uncertainty interval (e.g., single-digit ms in Google's original Spanner deployment) rather than a point-in-time accuracy figure; correctness follows from reasoning about the interval, not from the interval being small	External consistency for globally distributed control/metadata stores

Lamport clocks and Hybrid Logical Clocks are the right tool when causal ordering – not proximity to wall-clock time – is what is actually required, particularly across independently and imperfectly synchronized services (Sections 6.2–6.3). TrueTime-style uncertainty intervals are suited to systems that must reason explicitly about bounded clock uncertainty in distributed transactions, as in Spanner, rather than to general AI-cluster observability; none of the three constructs in this table should be read against Table 2's Evidence-Bounded Performance Statement column, since none of them make a comparable wall-clock performance claim.

9.3. Illustrative Measurement-Error Objectives by Workload

The values below are illustrative measurement-error objectives – useful as a starting point for design discussions – rather than derived requirements; they assume a synchronous, single-datacenter cluster and have not been validated against a specific production workload. Several rows describe where synchronization aids observability rather than where it is a correctness precondition; this is noted explicitly per row.

The table below separates one empirically reported result from the deployment-specific derivations required for the remaining functions. It does not assign generic numeric requirements. For each deployment, the objective should be derived from the duration and granularity of the event being resolved, the attribution error the use case can tolerate, the topology and hop count, the clock maximum time error, timestamp-capture error, and instrumentation overhead.

A correct derivation first requires separating several distinct error terms that this kind of illustrative discussion is prone to conflating – what this paper refers to as the four-term attribution-error decomposition: (1) clock synchronization error – how far apart two hosts' clock epochs actually are, which is what Table 2 characterizes per technology; (2) timestamp capture error – the jitter between an event occurring and a timestamp being recorded for it, which differs substantially between hardware capture at a NIC/switch ASIC and software capture in the kernel or user space; (3) software instrumentation latency – the delay a tracing or logging call itself adds to the code path being measured; and (4) event-processing/tracing overhead – the cost of buffering, serializing, transmitting, and correlating trace records after capture, which is a throughput and resource cost rather than a timestamp-accuracy term. These four terms have different sources, are not simply additive in general, and only the first is what Table 2 speaks to; conflating them risks presenting a single illustrative number as if it accounted for all four.

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

Cross-host tracing also does not require hardware timestamping at every hop: what matters is that the specific events being compared across hosts share a low enough combined error (terms 1 and 2 of the decomposition above) for their relative order and spacing to be trustworthy, not that every intermediate hop in the network path carries its own hardware timestamp. A pipeline that hardware-timestamps only at the measurement boundaries of interest, while relaying software-timestamped or untimed traffic in between, can still achieve tight cross-host attribution for the events it actually measures.

To derive a real bound for a specific deployment, a practitioner needs the duration and granularity of the event being resolved; the attribution error the use case can tolerate; the fabric topology and hop count, with wire serialization, switch transit, NIC-to-NIC, and application-visible RDMA latency reported separately; the chosen clock technology's validated maximum time error; the timestamp-capture error for the specific capture point; and the tracing and instrumentation overhead budget the workload can absorb. This checklist, rather than a generic threshold, is what this paper offers for deriving a deployment-specific bound. Table 4 therefore records the required derivation or evidence for each function instead of retaining unsourced numeric placeholders.

Table 4. Deployment-specific synchronization-objective derivation and evidence for common AI-infrastructure functions. Only the causal-ordering row reports a cited empirical observation; the remaining rows define what must be derived or validated.

AI Infrastructure Function	Deployment-Specific Derivation or Evidence	Role of Time Synchronization
Cross-host straggler/latency attribution during AllReduce	Derive from the shortest cross-host delay to resolve and the allowed attribution error; no general value supplied	Observability aid: enables attributing idle/wait time to a specific host or link; not a collective-correctness requirement
Checkpoint trigger-skew scheduling	Derive from the allowed trigger-skew budget for the checkpoint scheduler; no general value supplied	Observability/scheduling aid: bounds how far shard trigger times drift; checkpoint consistency itself is provided by the checkpoint protocol, not clock alignment
Cross-host distributed tracing	Derive from event spacing, expected clock uncertainty, and the ordering-ambiguity budget; no general value supplied	Observability aid: improves temporal placement of trace events; causal order itself comes from trace/span identifiers
Inference SLA / TTFT measurement	Derive from the per-stage SLO and the permitted latency-attribution error; no general value supplied	Observability aid: improves per-stage latency attribution across hosts
Causal-ordering robustness in Inference tracing	Reference [15] reports no observed violations through about 3 ms and violations by about 5 ms in one tested setup; not a general threshold	Empirically motivated: [15] reports no observed violations through ~3 ms and violations appearing by ~5 ms in one tested setup; actual threshold likely depends on workload timing, topology, and reconstruction algorithm
Rate limiting / quota enforcement	Reliable monotonic local clock; cross-host wall-clock synchronization is not required for correctness	Correctness depends on monotonic elapsed time plus authoritative shared state, not wall-clock synchronization
General log correlation / audit	Derive from reconciliation tolerance, log timestamp granularity, and audit policy; no general value supplied	Observability aid: supports manual reconciliation across logs

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

10. Open Challenges and Future Directions

10.1. Technical Open Challenges

- **Calibration – thermal- and load-induced drift:** GPU and NIC oscillators can drift under variable thermal load and power states typical of bursty AI workloads. Simulation studies of proposed high-accuracy PTP architectures suggest achievable sub-50 ns figures under idealized, modeled conditions [4], but that specific study does not evaluate GPU/NIC thermal drift under real AI workloads; characterizing drift-aware compensation for AI-specific thermal and power patterns remains an open, empirically under-studied area.
- **Calibration – domain crossing between compute and network clocks:** correlating GPU/accelerator event timestamps (often derived from an independent on-die or CUDA/CUPTI-domain clock) with NIC PHC timestamps introduces its own cross-domain calibration problem, distinct from network-level PTP synchronization itself. Section 3.5 and Figure 2 sketch the calibration links this depends on in practice – GPU-to-host via CUPTI and host-to-PHC via cross-timestamping – each needing periodic re-calibration; what remains open is a rigorous, implementation-level treatment of how frequently each link should be re-calibrated and how to bound the resulting compound error under real AI workload conditions, rather than the mechanism itself being wholly unknown.
- **Uncertainty propagation – agentic causality:** as agentic workflows increase the depth and asynchrony of cross-service call graphs, timestamp-based causal reconstruction benefits from principled uncertainty bounds (in the spirit of TrueTime) as a complement to trace-identifier-based causality, particularly for automated root-cause analysis and auditability [12, 15].
- **Scalable observability at cluster scale:** capturing and correlating hardware timestamps across tens of thousands of endpoints, without imposing unacceptable overhead on training or inference throughput, remains an active systems-engineering problem, motivating techniques such as sampling-aware trace propagation and compressed causal representations [9, 17].
- **Standardization across heterogeneous accelerator vendors:** as clusters increasingly mix GPU, TPU, and custom-ASIC accelerators from different vendors, consistent hardware-timestamping semantics and exposure (not just network-level PTP) would be necessary for uniform cross-vendor observability.

10.2. Timing Security Considerations

The time-distribution plane described in Section 7 (Figure 5) is itself an attack surface that this paper has otherwise treated as trusted; a deployment that relies on synchronized time for straggler attribution, latency accounting, or causal-trace correlation inherits whatever integrity guarantees – or lack of them – that plane provides. Three categories of threat are documented in the timing-security literature and apply directly to the architecture in Figure 5, though evaluating specific countermeasures for AI clusters is left as future work rather than addressed here.

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

- **GNSS spoofing and jamming at the primary reference:** an attacker able to transmit counterfeit satellite signals to a GNSS receiver can shift the entire hierarchy's notion of time at its root; documented countermeasures include multi-constellation and multi-receiver cross-checking, signal-quality monitoring, and holdover-based anomaly detection that flags a sudden jump against a free-running oscillator [51]. Figure 5's two fault-separated external references (Section 7) provide redundancy against a single receiver failing or being spoofed, but do not by themselves detect a coordinated spoof of both.
- **Malicious or misconfigured grandmaster election:** PTP's Best Master Clock Algorithm is a distributed election without built-in authentication in its base form; a compromised or misconfigured node can win the election and propagate false time. IEEE 1588-2008 Annex K defined an experimental symmetric-key security extension, but that annex is not the security annex of IEEE 1588-2019. IEEE 1588-2019 defines its security approach in Annex P, and IEEE 1588d-2023 adds GDOI key-management guidance and clarifications to Annex P and the associated security option [53]. Published analyses document attacks against unauthenticated BMCA election and limitations of earlier security mechanisms [49, 50]. This architecture should therefore identify the authentication TLV, key-management method, trust anchors, rollover procedure, and interoperability assumptions actually used rather than refer generically to Annex K.
- **On-path timestamp manipulation (delay attacks):** an attacker positioned on the network path between a TimeTransmitter and TimeReceiver can selectively delay or manipulate Sync, Delay_Req, or Delay_Resp messages to shift a victim's computed offset without needing to forge cryptographic authentication; this class of "delay attack" is not fully addressed by message-authentication mechanisms alone, since the messages themselves can remain authentic while their timing is manipulated in transit [49, 50]. Redundant grandmasters and paths (Figure 5) can support cross-checking a host's PTP-derived offset against an independent path, which is one documented network-based mitigation, though this paper does not evaluate its effectiveness for AI cluster fabrics specifically.

None of these threats are unique to AI infrastructure, but AI clusters concentrate a large amount of value behind a timing plane that this paper has argued is increasingly relied upon for operational diagnosis; a compromised time base would not only mislead an operator's straggler or latency attribution but could, in principle, be used to mask the very anomalies that observability tooling is meant to surface. A full timing-security threat model for AI-scale deployments – including which attacks matter most given typical AI-cluster trust boundaries and which mitigations are worth their operational cost – is left for future work.

11. Design Considerations for AI Temporal Infrastructure

The distinctions made throughout this paper translate into a small set of practical heuristics for architects and operators. These are offered as heuristics grounded in the paper's analysis, not as validated engineering rules with quantitative thresholds beyond the illustrative ones already qualified in Section 9.3.

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

- **Don't substitute synchronization for a consistency protocol:** wall-clock agreement is not a substitute for the protocol ordering, versioned state, and atomic commit that actually provide correctness for collectives, checkpoints, and distributed transactions (Table 1).
- **Treat GPU, CPU, NIC, and switch clocks as independent domains unless explicitly calibrated:** a PTP-synchronized NIC does not automatically make GPU-side event timestamps comparable across hosts (Section 3.5, Figure 2).
- **Use physical time for placement, not just precision:** synchronized wall-clock time is most valuable for comparing when events happened across hosts and domains – latency attribution, straggler localization – rather than as an end in itself.
- **Use logical time and trace context for causality:** Lamport clocks, Hybrid Logical Clocks, and trace/span identifiers establish correct ordering without depending on tight physical-clock agreement, and should be the default mechanism for causal reconstruction (Section 6.2).
- **Prefer hardware timestamps at boundaries when measured software-capture uncertainty exceeds the attribution-error budget:** software timestamping can be adequate for coarse correlation, but OS scheduling jitter and interrupt coalescing must be measured against the workload's attribution-error budget. Use hardware timestamping when the measured software-capture uncertainty exceeds that budget (Section 9.3).
- **Plan for recalibration, not a one-time calibration:** cross-domain offsets (GPU-to-host, host-to-PHC) drift independently under thermal and load variation typical of AI workloads; treat recalibration frequency as an open design parameter to validate for your own hardware and workload, not a solved problem (Section 10).
- **Derive precision from what the workload needs, not from the clock system's maximum capability:** picking the tightest available synchronization by default risks unnecessary engineering cost without a corresponding benefit; size the objective to the event-resolution the function actually requires (Section 9.3).
- **Label illustrative figures as illustrative:** when a synchronization bound or example scenario has not been measured against a real deployment, say so explicitly rather than letting a plausible-looking number be read as a validated requirement.
- **Treat the timing plane as part of the trust boundary, not outside it:** a deployment that relies on synchronized time for diagnosis should budget for grandmaster authentication, GNSS anomaly monitoring, and redundant/cross-checkable paths as part of its design, not as an afterthought (Section 10.2).

12. Conclusion

Precise time synchronization is foundational for cross-host measurement, one-way latency attribution, and functions explicitly built around bounded physical time or scheduled deadlines in AI distributed infrastructure. It is not, in general, a universal correctness prerequisite for NCCL-style collective communication, distributed checkpoint consistency, rate limiting, or causal tracing, which are normally governed by protocol ordering, logical progress, versioned state, and trace

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

relationships. Hardware-timestamped PTP, layered with software-level logical and hybrid logical clocks, provides the measurement accuracy and causal-reasoning support that make AI-scale bulk-synchronous training and low-latency, multi-tenant inference diagnosable and attributable at scale, even where it is not what makes them correct. Organizing this around explicit time-distribution, event-timestamping, and temporal-observability planes (Section 7) helps keep those roles from being conflated, and treating the time-distribution plane itself as part of the trust boundary (Section 10.2) rather than an assumed-trustworthy substrate is a further, often-overlooked requirement for relying on it operationally. As cluster scale, workload asynchrony, and agentic complexity continue to grow, the observability and measurement role of time synchronization deserves the same deliberate engineering attention as compute and network topology design – scoped accurately to what it does and does not provide.

13. Glossary

Term	Definition
AI	Artificial Intelligence
BMCA	Best Master Clock Algorithm
CPU	Central Processing Unit
GNSS	Global Navigation Satellite System
GPU	Graphics Processing Unit
HLC	Hybrid Logical Clock
NIC	Network Interface Card
NTP	Network Time Protocol
OCP	Open Compute Project
PHC	PTP Hardware Clock
PTP	Precision Time Protocol
SLA	Service-Level Agreement
SLO	Service-Level Objective
TAP	Open Compute Project Time Appliances Project
ToR	Top of Rack
TTFT	Time to First Token

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

14. References

References are grouped into (A) core time-synchronization and distributed-clock theory, (B) industry systems that report operationalizing software-based clock synchronization for AI fabrics (vendor claims, not independently validated), (C) AI training/inference systems literature, (D) accelerator-interconnect, collective-communication, and large-scale training-system literature, (E) timing security literature, and (F) data-center PTP profile and standards-amendment specifications, which together supply the operational context, workloads, failure modes, threat models, and concrete standards to which the time-synchronization principles in (A) and (B) are applied. Each entry notes specifically what it is cited for.

A. Time Synchronization and Distributed-Clock Foundations

- [1] IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, IEEE Std 1588-2019.
- [2] IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems – Amendment 2: Master-Slave Optional Alternative Terminology, IEEE Std 1588g-2022 (amendment to IEEE Std 1588-2019).
- [3] D. Mills, J. Martin (Ed.), J. Burbank, and W. Kasch, “Network Time Protocol Version 4: Protocol and Algorithms Specification,” RFC 5905, IETF, 2010; M. Lichvar and A. Malhotra, “NTP Interleaved Modes,” RFC 9769 (updates RFC 5905), IETF, May 2025, doi:10.17487/RFC9769, <https://www.rfc-editor.org/info/rfc9769>.
- [4] F. Li, W. Liu, Y. Qi, Q. Li, and G. Liu, “An Enhanced Method for Nanosecond Time Synchronization in IEEE 1588 Precision Time Protocol,” *Processes*, vol. 11, no. 5, article 1328, 2023, doi:10.3390/pr11051328.
- [5] K. J. Son and T. G. Chang, “Distributed Nodes-Based Collaborative Sustaining of Precision Clock Synchronization upon Master Clock Failure in IEEE 1588 System,” *Sensors*, vol. 20, no. 20, p. 5784, 2020.
- [6] J. Han and D.-K. Jeong, “A Practical Implementation of IEEE 1588-2008 Transparent Clock for Distributed Measurement and Control Systems,” *IEEE Transactions on Instrumentation and Measurement*, vol. 59, no. 2, pp. 433–439, 2010.
- [7] A. Mahmood, M. I. Ashraf, M. Gidlund, and J. Torsner, “Over-the-Air Time Synchronization for URLLC: Requirements, Challenges and Possible Enablers,” arXiv:1807.00078, 2018; also presented at IEEE Int. Symposium on Wireless Communication Systems (ISWCS), 2018.
- [8] Y. Geng, S. Liu, Z. Yin, A. Naik, B. Prabhakar, M. Rosenblum, and A. Vahdat, “Exploiting a Natural Network Effect for Scalable, Fine-grained Clock Synchronization,” in *Proc. 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI '18)*, pp. 81–94, 2018.
- [9] Y. Geng, S. Liu, Z. Yin, A. Naik, B. Prabhakar, M. Rosenblum, and A. Vahdat, “SIMON: A Simple and Scalable Method for Sensing, Inference and Measurement in Data Center Networks,” in *Proc. 16th USENIX Symposium on Networked Systems Design and Implementation (NSDI '19)*, pp. 549–564, 2019.
- [10] S. Liu, A. Ghalayini, M. Alizadeh, B. Prabhakar, M. Rosenblum, and A. Sivaraman, “Breaking the Transience-Equilibrium Nexus: A New Approach to Datacenter Packet Transport,” in *Proc. 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI '21)*, pp. 47–63, 2021.
- [11] S. Kulkarni, M. Demirbas, D. Madappa, B. Avva, and M. Leone, “Logical Physical Clocks and Consistent Snapshots in Globally Distributed Databases,” University at Buffalo Technical Report, 2014.
- [12] J. C. Corbett et al., “Spanner: Google’s Globally-Distributed Database,” in *Proc. 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2012.

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

- [13] L. Lamport, “Time, Clocks, and the Ordering of Events in a Distributed System,” *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.
- [14] B. H. Sigelman et al., “Dapper, a Large-Scale Distributed Systems Tracing Infrastructure,” Google Technical Report, 2010.
- [15] A. Sharma, D. Shah, D. Lariviere, and H. ElBakoury, “Time, Causality, and Observability Failures in Distributed AI Inference Systems,” arXiv:2604.21361v1, Apr. 23, 2026, doi:10.48550/arXiv.2604.21361.
- [16] I. Lagwankar and K. Wijewardena, “Causality Diagrams using Hybrid Vector Clocks,” arXiv:2311.07535, 2023.
- [17] V. Tekken Valapil, S. Kulkarni, E. Torng, and G. Appleton, “Efficient Two-Layered Monitor for Partially Synchronous Distributed Systems,” Technical Report, arXiv:2007.13030, 2020.

B. Industry Systems Reporting Software Clock Synchronization for AI Fabrics

These are vendor/industry sources, not peer-reviewed papers, and are identified explicitly as vendor claims rather than independently validated evidence throughout Sections 2 and 4.

- [18] Clockwork.io, “Closing the AI Networking Visibility Gap,” white paper, July 2025, clockwork.io/wp-content/uploads/2025/07/Clockwork.io-Closing-the-AI-Networking-Visibility-Gap.pdf (accessed Aug. 2026).
- [19] Clockwork.io, “About Us” (company technology overview describing Software-Driven AI Fabrics), clockwork.io/about-us/ (accessed Aug. 2026).
- [20] Clockwork.io, “The Research that Inspired Clockwork Systems,” company blog, clockwork.io/blog/the-research-that-inspired-clockwork-systems/ (accessed Aug. 2026).

C. AI Training and Inference Systems Literature (Application Context)

The following references do not concern time-synchronization protocols themselves; they are cited as evidence for the AI-infrastructure workloads, failure modes, and system designs discussed in Sections 3–6.

- [21] Z. Jiang et al., “MegaScale: Scaling Large Language Model Training to More Than 10,000 GPUs,” in *Proc. 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2024.
- [22] J. Lin et al., “Understanding Stragglers in Large Model Training Using What-if Analysis,” arXiv:2505.05713, 2025.
- [23] F. Liang, Z. Zhang, H. Lu, V. C. M. Leung, Y. Guo, and X. Hu, “Communication-Efficient Large-Scale Distributed Deep Learning: A Comprehensive Survey,” arXiv:2404.06114, 2024.
- [24] J. Duan et al., “Efficient Training of Large Language Models on Distributed Infrastructures: A Survey,” arXiv:2407.20018, 2024.
- [25] J. Cheng, N. Gao, Y. Yue, Z. Ye, J. Jiang, and J. Sha, “EDiT: A Local-SGD-Based Efficient Distributed Training Method for Large Language Models,” arXiv:2412.07210, 2024.

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

D. Accelerator Interconnects, Collective Communication, and Large-Scale Training Systems (Extended Bibliography)

Like Group C, these references establish infrastructure and workload context rather than time-synchronization mechanics, discussed in Sections 4.4, 6.1, and 6.2. IEEE 1588-2019 is already listed as reference [1] and is not repeated here.

[26] NVIDIA Corporation, “NVSwitch: The World’s Highest-Bandwidth On-Node Switch,” NVIDIA Technical Overview, images.nvidia.com/content/pdf/nvswitch-technical-overview.pdf; see also NVIDIA’s current NVLink/NVLink Switch product documentation at nvidia.com/en-us/data-center/nvlink/.

[27] UALink Consortium, “UALink 200G 1.0 Specification” (also distributed as “UALink_200 Rev 1.0 Specification”), ratified April 2025, ualinkconsortium.org – cited here as the historical original release; the Consortium’s current catalog also includes separate “UALink Common 2.0 Specification” and “UALink 200G Data Link and Physical Layers 2.0 Specification” documents, published 2026.

[28] Open Compute Project (OCP), “ESUN (Ethernet for Scale-Up Networking) – Network Operator Requirements – Base Specification Rev. 1.0,” released February 9, 2026.

[29] Ultra Ethernet Consortium (UEC), “Ultra Ethernet Specification,” Version 1.0.3, July 16, 2026, ultraethernet.org.

[30] A. Douillard et al., “DiLoCo: Distributed Low-Communication Training of Language Models,” arXiv:2311.08105, 2023.

[31] NVIDIA Corporation, “NCCL: NVIDIA Collective Communications Library,” Documentation and User Guide, 2024.

[32] M. Lipinski, T. Wlostowski, J. Serrano, and P. Alvarez, “White Rabbit: a PTP Application for Robust Sub-nanosecond Synchronization,” IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS), 2011, doi:10.1109/ISPCS.2011.6070148.

[33] A. Singh et al., “Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google’s Datacenter Network,” in Proc. ACM SIGCOMM 2015, pp. 183–197, doi:10.1145/2785956.2787508.

[34] Broadcom Inc., “Broadcom Ships Tomahawk 5, Industry’s Highest Bandwidth Switch Chip to Accelerate AI/ML Workloads,” Aug. 16, 2022, <https://www.broadcom.com/company/news/product-releases/60456> (accessed Aug. 21, 2026).

[35] AMD, “AMD Instinct MI300X Platform” data sheet, amd.com/en/products/accelerators/instinct/mi300/mi300x.html (product brief: amd.com/content/dam/amd/en/documents/partner-hub/instinct/instinct-mi300x-accelerator-product-briefs.pdf, 2023).

[36] E. Erdil, “Data Movement Bottlenecks to Large-Scale Model Training: Scaling Past 1e28 FLOP,” Epoch AI, Nov. 2024 (see also the accompanying technical report “Data Movement Limits to Frontier Model Training”).

[37] R. Thakur, R. Rabenseifner, and W. Gropp, “Optimization of Collective Communication Operations in MPICH,” International Journal of High Performance Computing Applications, 2005.

[38] Z. Zhang, C. Chang, H. Lin, Y. Wang, R. Arora, and X. Jin, “Is Network the Bottleneck of Distributed Training?,” in Proc. 2020 ACM Workshop on Network Meets AI & ML (NetAI ’20), pp. 8–13, 2020, doi:10.1145/3405671.3405810.

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

- [39] C. Renggli et al., “Communication-Efficient Deep Learning: A Survey,” *Foundations and Trends in Machine Learning*, 2023.
- [40] OpenAI, “GPT-4 Technical Report,” arXiv:2303.08774, 2023.
- [41] S. Rajbhandari et al., “ZeRO-Infinity: Breaking GPU Memory Wall for Extreme Scale Deep Learning,” *Proceedings of SC21*, 2021.
- [42] M. Shoeybi et al., “Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism,” arXiv:1909.08053, 2019.
- [43] D. Narayanan et al., “Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM,” *Proceedings of SC21*, 2021.
- [44] A. Douillard et al., “Decoupled DiLoCo for Resilient Distributed Pre-training,” arXiv:2604.21428v1, Apr. 23, 2026, doi:10.48550/arXiv.2604.21428.
- [45] A. Weingram et al., “xCCL: A Survey of Industry-Led Collective Communication Libraries for Deep Learning,” *Journal of Computer Science and Technology*, 2023.
- [46] S. Lee and J. Lee, “Collective Communication Performance Evaluation for Distributed Deep Learning Training,” *Applied Sciences*, vol. 14, no. 12, article 5100, 2024, doi:10.3390/app14125100.
- [47] G. Wang et al., “Blink: Fast and Generic Collectives for Distributed ML,” *MLSys*, 2020.
- [48] NVIDIA Corporation, “CUPTI Activity API,” CUPTI 12.8 Documentation, Section 5.1, https://docs.nvidia.com/cupti/12.8/api/group__CUPTI__ACTIVITY__API.html (accessed Aug. 21, 2026).

E. Timing Security Literature

These references support the timing-security discussion in Section 9.2; they establish that PTP and GNSS timing infrastructure have documented, actively studied attack surfaces, not that this paper proposes new defenses.

- [49] E. Itkin and A. Wool, “A Security Analysis and Revised Security Extension for the Precision Time Protocol,” *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 1, pp. 22–34, 2020, doi:10.1109/TDSC.2017.2748583 (originally arXiv:1603.00707).
- [50] W. Alghamdi and M. Schukat, “Precision Time Protocol Attack Strategies and Their Resistance to Existing Security Extensions,” *Cybersecurity*, vol. 4, no. 1, article 39, 2021.
- [51] D. Schmidt, K. Radke, S. Camtepe, E. Foo, and M. Ren, “A Survey and Analysis of the GNSS Spoofing Threat and Countermeasures,” *ACM Computing Surveys*, vol. 48, no. 4, article 64, 2016.

F. Data Center Profile and Security Amendments

These references define a concrete deployable profile and a formal standards amendment, rather than research analysis of attack surfaces, and are grouped separately from the timing-security literature in Group E for that reason.

- [52] Open Compute Project, “Data Center PTP Profile,” primary version 1, revision 1, Nov. 2022, <https://www.opencompute.org/documents/ocp-dc-ntp-profile-v1r1-pdf-2> (accessed Aug. 21, 2026).
- [53] IEEE Std 1588d-2023, IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, Amendment 4: GDOI (Group Domain of Interpretation) Key Management, 2023, <https://standards.ieee.org/ieee/1588d/11091/> (accessed Aug. 21, 2026).

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

15. License

15.1. Creative Commons Attribution-ShareAlike 4.0 International

OCP encourages participants to share their proposals, specifications and designs with the community. This is to promote openness and encourage continuous and open feedback. It is important to remember that by providing feedback for any such documents, whether in written or verbal form, that the contributor or the contributor's organization grants OCP and its members irrevocable right to use this feedback for any purpose without any further obligation.

It is acknowledged that any such documentation and any ancillary materials that are provided to OCP in connection with this document, including without limitation any white papers, articles, photographs, studies, diagrams, contact information (together, "Materials") are made available under the Creative Commons Attribution-ShareAlike 4.0 International License found here: <https://creativecommons.org/licenses/by-sa/4.0/>, or any later version, and without limiting the foregoing, OCP may make the Materials available under such terms.

As a contributor to this document, all members represent that they have the authority to grant the rights and licenses herein. They further represent and warrant that the Materials do not and will not violate the copyrights or misappropriate the trade secret rights of any third party, including without limitation rights in intellectual property. The contributor(s) also represent that, to the extent the Materials include materials protected by copyright or trade secret rights that are owned or created by any third-party, they have obtained permission for its use consistent with the foregoing. They will provide OCP evidence of such permission upon OCP's request. This document and any "Materials" are published on the respective project's wiki page and are open to the public in accordance with OCP's Bylaws and IP Policy. This can be found at <http://www.opencompute.org/participate/legal-documents/>. If you have any questions, please contact OCP.

Date: October 1, 2026



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

16. About Open Compute Foundation

The Open Compute Project (OCP) brings at-scale innovations and hyperscaler best practices to all, spanning technology domains from the data center to the edge, and the technology stack from silicon, to systems, to site facilities and services. The international OCP Community is made up of organizations and people from hyperscale and tier-2 cloud data center operators, communications providers, colocation providers, diverse enterprises, and technology vendors. The OCP Foundation fosters collaboration within the Community to tackle market challenges in an array of OCP Projects that create open contributions such as specifications, designs, and more. The OCP Solution Provider Program then recognizes the application of those contributions in compliant products, solutions, and data center facilities and services with designations like OCP Inspired, OCP Accepted and OCP Ready. With the tenets of openness, impact, efficiency, scale and sustainability, the OCP engages and educates thousands of engineers every year through many events and webinars. Across many initiatives the OCP Foundation and Community are meeting the market today and shaping the future.

Learn more at: www.opencompute.org.