Modular GPU Clouds with Optical Interconnect: A Specification of Placement, Exchange, and Recovery

Flyxion
Independent Researcher

November 2026

Abstract

Large data centers gather demand for power, land, and cooling at a small number of sites, and the heat they reject is seldom used where it is produced. A GPU cloud built from small modules placed where heat is already wanted, in places such as shopping malls and industrial plants, would invert that arrangement. It can do so only if the service says what a job may rely on when its modules are small, numerous, and available on the schedule of a host. This monograph specifies a GPU cloud whose basic unit of allocation is a fast local compute domain together with an explicit contract for exchanging state with other domains. The hardware is deliberately conventional: GPU servers or NVLink racks inside each module, and an optical Ethernet fabric between modules. The contribution lies in what the service promises about the boundary. A job requests a communication shape as well as a device count, the service reports the placement it can commit to, applications declare what must cross a module boundary and what makes it valid on arrival, and every committed checkpoint carries enough information for a replacement worker to resume under the job's declared recovery contract and for the customer to check the outcome. The treatment applies the distinction between local transitions and global admissibility to cloud scheduling. A sheaf over the job's partition complex specifies when local states can be assembled into a global state. An epoch can commit when the assembled state satisfies the declared global invariant and the proposal still names the current parent checkpoint. A fluid scheduling model determines whether the required boundary exchanges can arrive by their deadlines. Two applications, hierarchical synchronization for distributed training and context-aware assembly for parallel transcription, serve as test cases, and a two-module prototype is described together with the measurements that should govern any later decision to buy faster optics. No performance results are claimed here; the document fixes definitions, requirements, and experimental protocols.

Contents

I Foundations **4**

1 Introduction **5**

2 Local Transitions and Global Admissibility **7**
2.1 Joint systems and boundary decomposition 7
2.2 Drift between exchanges 8

3 Hardware Scopes and Their Bandwidths **9**

II Specification **11**

4 Modules and Placement **12**
4.1 An illustrative request and report 13

5 Exchange Contracts **14**
5.1 Refinement of contracts 15

6 Checkpoints and Recovery **16**

7 Result Records **17**

8 Sheaf Semantics of Distributed State **18**
8.1 The partition complex and the state sheaf 18
8.2 Global invariants as evidence 19
8.3 The obstruction to gluing 19

9 Typed Commit of an Epoch **21**

10 Feasibility of Optical Exchange **23**
10.1 A diagnostic for contended links 24

11 A Worked Example in Three Attempts **25**
11.1 Setup 25
11.2 Attempt A: a defect outside the image 26
11.3 Attempt B: a glued state that exceeds the budget 27
11.4 Attempt C: a state that glues and meets the budget 27
11.5 Summary of the attempts 29
11.6 Counting attempts 29

III Applications **31**

12 Synchronization and Staleness **32**
12.1 The shared shard and its history 32
12.2 A locally valid gradient withheld or revised 33

13 Context-Aware Assembly and Unresolved Names **36**
13.1 Mentions, evidence, and the transcript invariant 36
13.2 Overlaps and the discrete repair class 37
13.3 A worked case with four commit attempts 37
13.4 The delivered record 39

IV Evaluation and Procurement **40**

14 Experimental Protocols **41**
14.1 Common apparatus and accounting 41
14.2 Contract runs 42
14.3 Training runs 42
14.4 Fabric runs 44
14.5 Transcription runs 45
14.6 Reporting 46
14.7 What the protocols cannot establish 46

15 The Procurement Decision **47**

A Conformance Clauses **49**
A.1 Stating a conformance claim 52

B Requirement Traceability **53**

Part I

Foundations

Chapter 1

Introduction

A GPU cloud sells devices, and the devices are fast. What it rarely sells is a statement about how a computation spread across several devices remains a single computation. Two modules can each run correctly in isolation while the combined result is wrong at the seam between them, because the information that had to cross the seam was late, stale, truncated, or never specified. Optical interconnect enlarges the amount of information that can cross a seam per unit time. It does not say which information must cross, how often, or what makes it acceptable on arrival. Those are properties of the application and of the service contract, and they are the subject of this monograph.

The question of what a seam promises has become practical because of where computation is being placed. The International Energy Agency projects that electricity demand from data centers worldwide will more than double by 2030, to around 945 TWh, with AI the largest single driver [6]. Reporting on the same analysis notes that without investment in transmission, up to a fifth of planned data center projects could be delayed [7]. The load gathers at the few sites where grid capacity can be had, and those sites then become load centers that their neighbors contest. One tracker of local opposition counted twenty projects delayed or blocked in the second quarter of 2025, representing an estimated $98 billion of potential investment [8]. The pattern resembles urban concentration in a plain sense: a resource attracts a dense settlement, and the settlement's costs in land, power, water, and heat fall on the surrounding community.

Heat is the cost with the most obvious remedy. Almost all of the electrical energy a GPU consumes leaves the building as heat, and at a large data center that heat is usually rejected at a temperature too low for a district heating network to use without a heat pump [10]. Reuse projects exist, for example in Finland and near Copenhagen [8], but they meet financial and technical obstacles, and in one Danish case councillors stalled the planning process while seeking assurances about reuse, the first data center was approved while reuse remained uncertain, and the local heating utility's plan covered the heat of one of the five proposed facilities [9]. The heat is produced in one place and wanted in another.

The inverse arrangement puts small GPU clusters where heat is already bought, in shopping malls, in industrial plants with space or process heating, and in similar buildings. The electricity is spent in either case, so computation at such a site displaces part of a heating load and does not add a cooling load beside it. The idea has precedent. A French company has placed servers inside radiator-like heaters and building boilers, and rents the computing capacity of the resulting network to customers [10, 11]. It also has an evident limit, which is that heat demand is local and seasonal. A cluster in a mall can use its full power in January and may be asked to draw little in July. Its availability follows the heat demand of its host and not the demand of its customer.

A cloud made of such modules cannot sell a uniform pool of devices. Its modules are small, numerous, joined by ordinary or leased fiber and not by a rack fabric, and available on a schedule that someone else sets. What such a service must say is what shape of job fits, what crosses between sites and how often, what happens when a site stops taking power, and what is committed at each boundary. Those are the questions this monograph specifies. Tightly coupled training remains in rack-scale domains, where the interconnect of Chapter 3 applies. The distributed case is the loosely coupled and recoverable one, in which the exchange contracts, the recovery contract, and the commit protocol of the later chapters carry the weight. The monograph makes no claim about the economics or the thermal engineering of host sites; site attributes enter only as inputs to placement.

The design proposed here has three commitments. The first is that the unit of allocation is a module, meaning a fast local domain with a known internal interconnect, and that the scheduler exposes the topology that produces any bandwidth figure it advertises. The second is that scale-out between modules uses shipping optical Ethernet, with pluggable optics for the prototype and co-packaged optics at the switch as a production option to be evaluated rather than assumed. The third is that novelty is placed in software and contract: job placement, the crossing of module boundaries by applications, and the checking that a job can continue after a link, worker, or module has failed.

The document is organized as a monograph with a specification at its center. Part I fixes vocabulary and sets out the hardware scopes involved, using figures from vendor documentation and industry reporting with their sources. Part II states the specification as numbered requirements: the module and placement model, exchange contracts, the checkpoint and recovery contract, result records, a sheaf semantics for distributed state, a typed commit of each epoch, and a feasibility condition for optical exchange, followed by a worked example in which commit attempts from one head end in each of the three rejection classes and in a receipt. Part III develops the two applications. Part IV gives the prototype, the experiments, and the bounds that connect their measurements to the procurement decision. The appendices collect the requirements into conformance tests and trace each requirement to its formal object, its record, its test, and the run that exercises it.

Chapter 2

Local Transitions and Global Admissibility in Compute Fabrics

Let a distributed computation be described by a finite family of modules $\mathcal{M}_1, \dots, \mathcal{M}_k$. Each module $\mathcal{M}_i$ has a local state space S_i and a local transition relation $\tau_i \subseteq S_i \times S_i$ describing the steps it may take without communication. A boundary between $\mathcal{M}_i$ and $\mathcal{M}_j$ is described by an exchange alphabet E_{ij}, a projection $\pi_{ij} \colon S_i \to E_{ij}$ saying what module i exposes, and a reception relation saying how module j incorporates what it receives.

Definition 2.1 (Local admissibility)**.** *A local transition $(s, s') \in \tau_i$ is locally admissible when s' satisfies every invariant that $\mathcal{M}_i$ can check from S_i alone.*

Definition 2.2 (Global admissibility)**.** *A joint state $(s_1, \dots, s_k)$ is globally admissible with respect to a predicate* Adm *on $S_1 \times \cdots \times S_k$ when* Adm$(s_1, \dots, s_k)$ *holds. A job is globally admissible at a synchronization step when the joint state it has reached at that step is globally admissible.*

The gap between the two definitions is the central fact. A sequence of locally admissible transitions in every module need not yield a globally admissible joint state, because Adm may constrain relations between components that no single module can inspect. Exchange is the mechanism that closes the gap, and an exchange contract is the statement of how much closing is promised. The interconnect determines the cost of exchange; it does not determine what exchange is sufficient. A faster fabric therefore moves the point at which a given contract becomes affordable, and leaves the contract itself to be written by someone.

Remark 2.1. The vocabulary here follows the author's earlier treatment of local transitions and global admissibility [1]. Nothing in this monograph depends on results from that treatment beyond the two definitions above.

2.1 Joint systems and boundary decomposition

The joint system has state space $S = S_1 \times \cdots \times S_k$. A step of the joint system is either a local step of a single module, or an exchange step at a boundary, in which the projection $\pi_{ij}(s_i)$ is transmitted to module j and a validator is evaluated on arrival. A synchronization schedule σ assigns to each boundary a set of step indices $T_{ij} \subseteq \mathbb{N}$ at which its exchange steps occur.

Definition 2.3 (Boundary decomposability)**.** *A predicate* Adm *is boundary decomposable when there exist local predicates* Loc_i *on S_i and boundary predicates R_{ij} on $E_{ij} \times E_{ji}$ such*

that

$$\mathsf{Adm}(s_1, \dots, s_k) \iff \bigwedge_i \mathrm{Loc}_i(s_i) \ \wedge \ \bigwedge_{i<j} R_{ij}\big(\pi_{ij}(s_i), \pi_{ji}(s_j)\big).$$

Proposition 2.1. *Suppose* Adm *is boundary decomposable. If at step t every module satisfies* Loc_i*, and every boundary has just completed an exchange whose validator evaluated R_{ij} on the current projections and succeeded, then the joint state at t is globally admissible.*

Proof. Every conjunct in the decomposition of Adm has been established at step t. □

The proposition is immediate, and its weight lies in the hypothesis. A workload whose admissibility predicate is boundary decomposable can be certified by pairwise exchange and validation. A workload whose predicate is not decomposable cannot, whatever the interconnect.

Proposition 2.2. *There is a predicate* Adm *on three modules that is not boundary decomposable, and for which pairwise exchange and validation can accept an inadmissible joint state.*

Proof. Let each $S_i = \{0, 1\}$ with π_{ij} the identity, and let $\mathsf{Adm}(s_1, s_2, s_3)$ hold exactly when $s_1 \oplus s_2 \oplus s_3 = 0$. Any pair of values (s_i, s_j) extends to an admissible triple by choosing the third bit to fix the parity. A pairwise validator R_{ij} that accepts every pair occurring in some admissible state therefore accepts all four pairs. The triple $(1, 0, 0)$ passes all three pairwise validators and is inadmissible, so no choice of local and pairwise predicates reproduces Adm. □

The example is small by design. It shows that sufficiency of exchange is a property of the predicate and of the order of the relations a validator can check, not of the bandwidth available. Where the predicate involves relations among three or more modules, the contract must either exchange higher-order summaries or route validation through a module that receives all the projections involved. Chapter 8 gives the cells that carry such summaries.

2.2 Drift between exchanges

Between exchanges each module continues to take local steps, and the copy of its projection held by its neighbour grows stale. Suppose E_{ij} carries a metric d, and that a single local step of module i changes $\pi_{ij}(s_i)$ by at most δ_i in d.

Proposition 2.3. *If module j last received $\pi_{ij}(s_i)$ at step t_0, then after m local steps of module i the distance between the held copy and the true projection is at most $m\delta_i$. A tolerance ε_{ij} on this distance is therefore respected whenever $m \le \lfloor \varepsilon_{ij}/\delta_i \rfloor$.*

Proof. By the triangle inequality the total displacement over m steps is at most the sum of the m single-step displacements, each bounded by δ_i. □

The bound is conservative, since steps may partly cancel, but it is computable from a declared per-step bound and gives the scheduler a principled maximum interval for a contract. It also identifies what an application must supply: a metric on the exchanged state and a per-step bound. Applications that cannot supply them may still declare an interval, but the service can then certify nothing about drift and the outcome class of Chapter 6 should be chosen accordingly.

Chapter 3

Hardware Scopes and Their Bandwidths

Bandwidth figures in vendor material describe different scopes, and quoting them side by side invites a category error. This chapter separates the scopes that the specification refers to and records the figures with their sources. The figures are inputs to placement policy, not claims about delivered application performance.

Inside a rack-scale NVLink domain, the NVIDIA GB300 NVL72 connects 72 Blackwell Ultra GPUs and 36 Grace CPUs as one NVLink domain with a quoted aggregate NVLink bandwidth of 130 TB/s [2]. The reference architecture documentation describes nine NVLink switch trays with two switch ASICs each, and one NVLink link from each GPU to each in-rack switch over a copper backplane, forming a fully connected domain of 72 GPUs [3]. This is the largest single placement domain considered here.

Between modules, the relevant interface is the network adapter. The NVIDIA ConnectX-8 SuperNIC is documented at a maximum bandwidth of 800 Gb/s. Its datasheet lists InfiniBand operation at up to 800 Gb/s on one port and Ethernet speeds of 400, 200, 100, 50, and 25 Gb/s, with two-port configurations at 400 Gb/s each [4]. For an Ethernet scale-out fabric the specification therefore treats 800 Gb/s as an aggregate across two 400 Gb/s ports unless a particular deployment demonstrates otherwise. At the raw line rate, 800 Gb/s corresponds to 100 GB/s before protocol overhead.

At the switch, TrendForce reported on 27 July 2026 that NVIDIA had begun shipping its Spectrum-X co-packaged optical switch to select partners, with up to 400 Tb/s of switching capacity, and that Broadcom was continuing limited shipments of its 51.2 Tb/s Bailly switch, which the report says reduces power consumption by up to 70 percent relative to conventional pluggable transceivers [5]. The same report identifies optical engine yield, silicon photonics capacity, and advanced packaging as supply bottlenecks. Switch-level photonics is therefore a credible procurement path for a module fabric, with availability and serviceability to be verified for any specific deployment.

The scopes in Table 3.1 are drawn from one vendor's platforms, and the module model does not depend on that vendor. Huawei's published description of its CloudMatrix384 supernode has the same two-tier shape. Within the supernode, 384 Ascend 910C NPUs and 192 Kunpeng CPUs are joined by a Unified Bus. Between supernodes there is an RDMA plane, and compute instances inside a supernode are allocated with attention to topology [12]. In the vocabulary of Chapter 4 such a supernode is a module of rack scope and its inter-supernode plane is the boundary. Research on the optical side points the same way. A group at Peking University reports a distributed inference system built on an on-chip all-optical supernode, using a 400 Gbps silicon photonic transceiver chip and a 16×16

Scope	Example	Quoted figure
Rack NVLink domain	GB300 NVL72, 72 GPUs	130 TB/s aggregate
Module to fabric	ConnectX-8 SuperNIC	800 Gb/s maximum aggregate
Switch capacity	Spectrum-X CPO switch	up to 400 Tb/s
Switch capacity	Broadcom Bailly CPO switch	51.2 Tb/s

Table 3.1: Scopes referred to by the specification. The figures are not comparable with one another and are listed to fix what each placement domain exposes.

optical switch chip [13], and a separate paper from the same institution demonstrates 400 Gb/s on a single optical lane in a laboratory experiment [14]. These are laboratory results. The specification treats none of their figures as capacity that an operator can commit, since reservable capacity under Requirement 10.1 is whatever the operator is prepared to reserve.

Part II

Specification

Chapter 4

Modules and Placement

Definition 4.1 (Module)**.** *A module is a set of GPUs together with local memory, host memory, and storage staging that share a local interconnect and fail, or are repaired, as a unit for the purpose of placement. A module is described by its scope (single server, multi-server NVLink domain, or rack), its GPU count, its internal interconnect figures, and its external attachment to the fabric.*

Remark 4.1 (Host sites). A module may sit at a site whose purpose is heat and not computation, such as a mall or an industrial plant. The module's description then includes site attributes that the scheduler takes as placement inputs and does not infer: the power the site will let the module draw, and the periods in which it will, which follow the heat demand of the site. A period in which the site takes no power is a period in which the module is unavailable, and the constrained replacement requirement and the recovery contract of Chapter 6 govern what happens to a job placed there. Heat-following behavior therefore cannot change the admissibility of a job without a record. The specification fixes nothing about the economics or the thermal engineering of such sites.

Definition 4.2 (Communication shape)**.** *A communication shape is a request of the form* $(n_1, \ldots, n_k; \sigma)$ *asking for* k *modules, the* i*th containing* n_i *GPUs inside one fast domain, with a synchronization schedule* σ *describing the exchange intervals between modules. A shape with* $k = 1$ *is a tightly coupled request; a shape with* $k > 1$ *states how loosely the modules may be coupled.*

Requirement 4.1 (Shape-first allocation)**.** *A job request shall state a communication shape. A request stating only a device count shall be interpreted as the shape in which every device may be placed in a different module with no committed exchange schedule.*

Requirement 4.2 (Reported placement)**.** *The service shall return the actual placement as a list of modules, each with its scope, GPU count, and the internal and external interconnect figures the operator is prepared to commit to. Figures that describe a different scope from the one allocated, such as an aggregate rack figure for a job placed on part of a rack, shall not be reported as committed limits.*

Requirement 4.3 (Constrained replacement)**.** *When a module fails, the scheduler may replace it only with a module whose placement satisfies the shape of the original request, or with a placement the application has declared acceptable. If no such module exists the job shall be suspended at its last committed boundary and the customer notified, rather than resumed on a placement that changes its admissibility.*

Proposition 4.1. *Under the constrained replacement requirement, a job that resumes after a module failure resumes on a placement satisfying its original shape or an application-declared alternative.*

Proof. The scheduler has two permitted actions on failure, replacement under the stated condition and suspension. Resumption occurs only after a replacement action, whose condition is exactly the conclusion. □

4.1 An illustrative request and report

Consider a job that asks for two modules of eight GPUs each, joined by an exchange interval of m steps. Under Definition 4.2 this is the shape $(8, 8; \sigma)$, and the service answers with a placement report. The report names the module that holds the first eight GPUs, states its scope as a single server, and gives the internal interconnect figure the operator is prepared to commit to for that scope. It names the second module in the same way, possibly at another site, and gives the external attachment of each module to the fabric, with the reservable capacity and its basis in the sense of Requirement 10.1. It does not give an aggregate figure for the rack that happens to contain a module, because the job holds part of that rack and the rack figure describes a scope the job was not allocated.

A request that states only sixteen devices is read by Requirement 4.1 as a different shape, in which every device may sit in a different module and no exchange schedule is committed. Such a job receives the same kind of report, and nothing in the report suggests that its devices share a fast domain. The two requests therefore differ in what the service promises and not only in how the devices are counted. The first can be refused at admission if the fabric cannot carry its declared exchanges, by Requirement 5.4. The second cannot be refused on that ground, since it declared none, and it cannot later complain of a boundary it never described.

The same report shows what happens on failure. If the second module fails and a free module satisfies the shape, the scheduler replaces it and the new report states the new module’s scope and figures. If no such module is free, Requirement 4.3 suspends the job at its last committed boundary and informs the customer, and the job does not resume on a larger, slower, or more distant module that would change which exchanges can arrive on time. When the second module sits at a host site whose power is withdrawn for part of the year, as in Remark 4.1, the same rule applies: the withdrawal is a period in which the module is unavailable, and the job is replaced or suspended by the rule above and is not left to run in a degraded state under its original report.

Chapter 5

Exchange Contracts

An exchange contract is the application's statement of what crosses a module boundary. It is part of the job description and is checked by the service, not left to the application's internal conventions.

Definition 5.1 (Exchange contract)**.** *An exchange contract for the boundary between modules i and j consists of a schema for the exchanged state (the alphabet E_{ij} and its encoding), an interval stated in steps or in elapsed time, a validity predicate that the receiving side evaluates on arrival, and a tolerance specifying what deviation from the declared interval or content the job accepts.*

Requirement 5.1 (Declared exchange)**.** *Every cross-module transfer performed by a job shall be covered by an exchange contract. Transfers not covered by a contract shall be refused by the service's communication layer or logged as unsanctioned.*

Requirement 5.2 (Validation on arrival)**.** *The service shall record, for each exchange, whether the validity predicate held on arrival, together with the step at which it was evaluated. A failed validation shall be reported to the job and shall not be silently retried more times than the contract permits.*

Requirement 5.3 (Interval as a parameter)**.** *The exchange interval shall be a declared parameter of the job and shall be varied only by an application-level decision recorded in the job history. The service shall not change the interval to compensate for fabric conditions.*

The last requirement matters for evaluation. If the service could quietly lengthen an interval under congestion, convergence differences between runs would be attributable to an undeclared algorithmic change. Keeping the interval as a recorded parameter keeps it an experimental variable.

Requirement 5.4 (Feasibility at admission)**.** *Before accepting a job the service shall check that the committed placement can carry each declared exchange within its interval. Writing b_{ij} for the size of the exchanged state, B_{ij} for the committed bandwidth of the path between the modules in matching units, λ_{ij} for its committed latency, m_{ij} for the interval in steps, and τ_i for the measured or declared step time of module i, the service shall refuse the job when*

$$\frac{b_{ij}}{B_{ij}} + \lambda_{ij} \;>\; m_{ij}\,\tau_i$$

and the exchange is declared to be non-overlapping with computation. For an exchange declared to overlap with computation the same inequality is a necessary condition only, and the service shall report that the test is weaker.

The inequality is deliberately elementary. Its purpose is to make the relation between the fabric and the contract explicit before any compute is spent, and to produce a refusal that names the boundary and the quantity that failed.

5.1 Refinement of contracts

Definition 5.2 (Contract tuple and refinement)**.** *An exchange contract is written* $C_{ij} = (E_{ij}, T_{ij}, V_{ij}, \varepsilon_{ij})$ *with alphabet, exchange steps, validator, and tolerance. A contract* C'_{ij} *refines* C_{ij} *when it has the same alphabet,* $T'_{ij} \supseteq T_{ij}$*, the validator* V'_{ij} *implies* V_{ij}*, and* $\varepsilon'_{ij} \leq \varepsilon_{ij}$*.*

Proposition 5.1. *Any admissibility conclusion obtained from Propositions 2.1 and 2.3 under a contract* C_{ij} *remains valid under any refinement* C'_{ij}*.*

Proof. The conclusions use three facts about the contract: that exchanges occur at the steps of T_{ij}, that the validator holds on arrival, and that the drift between exchanges stays within the tolerance. Under the refinement, exchanges occur at least at those steps, so intervals between exchanges are no longer; the validator implies the original one; and the tolerance is no larger. Each fact therefore continues to hold. □

Refinement gives the scheduler a safe direction of substitution. When a replacement placement offers a faster path than the original, the job may be continued under a refined contract without re-establishing its guarantees. The reverse direction, relaxing a contract to fit a slower replacement, is a change of contract and requires the application-level decision recorded under the interval requirement.

Chapter 6

Checkpoints and Recovery

Definition 6.1 (Committed checkpoint)**.** *A committed checkpoint is a record identifying the input versions, the partition of work among workers, the software and runtime versions, and the synchronization step that the saved state represents, together with the saved state or a reference to it.*

Requirement 6.1 (Identified boundaries)**.** *A job shall commit checkpoints only at synchronization boundaries declared in its exchange contracts, so that every checkpoint corresponds to a defined joint state.*

Requirement 6.2 (Resume from boundary)**.** *After a worker or module failure the replacement shall resume from the last committed boundary and shall regenerate unfinished work. Work completed after that boundary and not committed shall be treated as lost.*

Requirement 6.3 (Outcome class)**.** *Each job shall declare an outcome class at submission. For a deterministic workload under a supported runtime, the class is matching output hashes between an uninterrupted run and an interrupted run. For training, the class names the checkpoint state compared and the tolerances applied. The service shall report which class the delivered result satisfies.*

Define the replayed fraction of an interrupted run as

$$\rho \;=\; \frac{W_{\text{after resume}} - W_{\text{uninterrupted remainder}}}{W_{\text{total}}},$$

where W denotes measured work in a common unit such as GPU-seconds. The replayed fraction is reported for every interruption experiment in Part IV, because it prices the checkpoint interval directly.

Chapter 7

Result Records

A completed job returns a result together with a record from which a customer can check the result. The record is the evidentiary counterpart of the checkpoint: the checkpoint supports continuation, and the record supports inspection.

Requirement 7.1 (Preserved candidates)**.** *Where a post-processing stage modifies a result produced by an earlier stage, the record shall preserve the original candidate and the span of input it was derived from.*

Requirement 7.2 (Evidence for correction)**.** *Each modification shall be accompanied by the evidence on which it rests, expressed in terms the customer can examine, such as the surrounding input segment and the rule or model component that proposed the change.*

Requirement 7.3 (Unresolved status)**.** *When the available evidence does not suffice to select a correction, the record shall mark the item unresolved and retain all candidates. The service shall not select among candidates by a rule that is not recorded.*

Chapter 8

Sheaf Semantics of Distributed State

Chapter 2 defined admissibility over a product of local state spaces and showed that pairwise exchange can be insufficient. This chapter supplies a structure in which interfaces of any arity are cells of a space, and in which the statement that modules agree on what they share becomes a sheaf condition. The structure is used by the commit protocol of Chapter 9 and by the feasibility test of Chapter 10.

8.1 The partition complex and the state sheaf

Definition 8.1 (Partition complex)**.** *The partition complex of a job is a finite poset X whose elements, called cells, are the modules of the placement, the pairwise interfaces between modules that share state, and the higher-order interfaces wherever several modules share a constraint. The order is incidence: each interface cell lies above every module cell it involves, and each higher-order interface cell lies above the pairwise cells it contains.*

Give X the Alexandrov topology, whose open sets are the up-sets. For a module i let U_i be the smallest open set containing its cell, which is the module cell together with every interface cell involving it. The intersection $U_i \cap U_j$ then consists of the interface cells that involve both modules, and the sets U_i cover X.

Definition 8.2 (State sheaf)**.** *For each logical epoch t the state sheaf $\mathcal{S}_t$ assigns to each open set U the set $\mathcal{S}_t(U)$ of locally admissible partial states on U. For opens $V \subseteq U$ the restriction map $\mathcal{S}_t(U) \to \mathcal{S}_t(V)$ forgets the data on cells of U outside V. In particular, restricting from a module neighbourhood to an overlap forgets private state and retains the data visible on the interface.*

A global state at epoch t is an element of $\mathcal{S}_t(X)$. By the sheaf condition for the cover $\{U_i\}$ it is the same thing as a family of local states that agree on overlaps:

$$\mathcal{S}_t(X) \;\cong\; \mathrm{Eq}\Big(\prod_i \mathcal{S}_t(U_i) \;\underset{d_1}{\overset{d_0}{\rightrightarrows}}\; \prod_{i<j} \mathcal{S}_t(U_i \cap U_j)\Big),$$

where $(d_0 s)_{ij} = s_i|_{U_i \cap U_j}$ and $(d_1 s)_{ij} = s_j|_{U_i \cap U_j}$.

What the interface carries decides what agreement means. An interface may hold boundary fluxes, an epoch number, or the digest of a replicated parameter block. If two modules publish only the norms of their gradients, agreement of the norms says nothing about agreement of the gradients, and the sheaf enforces exactly the compatibility its restrictions express and no more.

Requirement 8.1 (Declared interface content)**.** *For every interface cell the job shall declare the data it carries and the agreement relation on that data. Where an interface carries only a summary such as a norm or a digest, the declaration shall state that agreement of the summaries does not imply agreement of the underlying data.*

Remark 8.1. The sheaf supplies the remedy suggested after Proposition 2.2. In that example a triple interface cell whose stalk carries the three bits makes the parity predicate checkable on a single cell, where pairwise cells could not check it. Higher-order constraints are handled by adding higher-order cells to the partition complex, and the price is the cost of exchanging the data those cells carry.

8.2 Global invariants as evidence

Some constraints, such as a global power budget or conservation of total mass, do not decompose into conditions on overlaps. They are kept as an explicit obligation Inv_t on global sections, and admissibility is the dependent sum

$$\mathrm{Admissible}_t \;=\; \sum_{s : \mathcal{S}_t(X)} \mathrm{Inv}_t(s).$$

An admissible state is a pair: the assembled state together with evidence that the invariant holds of it. This separates two statements that are easily conflated, namely that the modules agree on what they share and that the resulting job state satisfies the contract.

Requirement 8.2 (Explicit global invariants)**.** *Any invariant that is not expressed as agreement on overlaps shall be declared as part of the job contract, and its evidence shall be produced at each commit.*

8.3 The obstruction to gluing

Let module i propose the next local state $s_i' = F_i(s_i)$. Assume an additive interface representation, so that each interface ij carries values in an abelian group I_{ij} and restriction from module i is a homomorphism $r_{i,ij}$. The defect on the overlap is

$$\delta_{ij} \;=\; r_{i,ij}(s_i') - r_{j,ij}(s_j').$$

Suppose module i is authorized to apply repairs $a_i \in A_i$, where A_i is a vector space encoding what the module may change, and that a repair alters its interface values by the linear map $L_{i,ij}$. Define the repair operator

$$(Da)_{ij} \;=\; L_{i,ij}\, a_i - L_{j,ij}\, a_j.$$

Proposition 8.1. *In the linear repair model, applying repairs a changes the defect from δ to $\delta + Da$. A compatible family is therefore reachable by permitted repairs if and only if $-\delta \in \operatorname{im} D$, equivalently $\delta \in \operatorname{im} D$, equivalently $[\delta] = 0$ in $\operatorname{coker} D$. When the spaces are finite dimensional the condition is the rank equality* $\operatorname{rank}[\, D \mid \delta \,] = \operatorname{rank} D$.

Proof. Compatibility on every overlap means that the repaired defect vanishes, that is $\delta + Da = 0$ for some a. Since $\operatorname{im} D$ is a subspace it contains $-\delta$ exactly when it contains δ. The rank statement is the standard criterion for solvability of a linear system. □

The class $[\delta]$ is a concrete obstruction: it names the mismatches that cannot be eliminated by the repairs currently available to the modules.

Proposition 8.2. *The image of D is determined by the authorized repair spaces A_i and the maps $L_{i,ij}$. It does not depend on the capacity, latency, or schedule of the interconnect.*

Proof. D is defined from A_i and $L_{i,ij}$ alone, and none of these refers to the fabric. □

Optical bandwidth can therefore help the system discover and exchange δ sooner, but it cannot enlarge $\operatorname{im} D$. If a boundary mismatch can be repaired only by revising a parameter that the job contract holds fixed, a faster switch cannot make the transition admissible. The remedies are to enlarge the authorized repairs, to revise the contract, or to change the partition so that the interface structure changes. When the repair spaces form a coefficient system over the partition complex the cokernel is the first cohomology of the two-term complex $\bigoplus_i A_i \to \bigoplus_{i<j} I_{ij}$ defined by D, which can be read as a ech class; the rank formulation above is already a checkable condition for the prototype.

Requirement 8.3 (Obstruction reporting)**.** *When a proposed epoch fails to glue, the service shall report which overlaps carry a nonzero defect and whether the defect lies in the image of the declared repair operator. A defect outside the image shall be reported as unrepairable under the current contract and shall not be resubmitted unchanged by the scheduler.*

Chapter 9

Typed Commit of an Epoch

The control plane can make the global commit an operation that consumes exactly the obligations of Chapter 8. Let $P_i(t, h)$ be the type of a proposal by module i based on epoch t and parent checkpoint digest h, containing its proposed next local state and the interface data it exposes. Let $\mathrm{Compat}(p)$ be the type of evidence that the overlaps of the proposed states agree, and let $\mathrm{glue}(p, c)$ construct the global section from proposals p and compatibility evidence c. Define

$$\mathrm{Ready}(t, h) := \sum_{p : \prod_i P_i(t,h)} \sum_{c : \mathrm{Compat}(p)} \mathrm{Inv}_{t+1}\big(\mathrm{glue}(p, c)\big),$$

$$\mathrm{commit}_{J,t,h} : \mathrm{Ready}(t, h) \longrightarrow \mathrm{IO}\big(\mathrm{Receipt}(J, t+1) + \mathrm{StaleParent}(J, t, h)\big).$$

An inhabitant of $\mathrm{Ready}(t, h)$ is a set of proposals, a proof that they glue, and a proof that the glued state satisfies the invariant of the next epoch. The type ties the evidence to the exact transition proposed, because every component mentions t and h.

A proposal can be valid when produced and stale when received. Validity of the evidence says nothing about whether the job has moved on, so the commit performs an atomic compare-and-swap of the head of job J against (t, h). The evidence is checked first and the swap decides which valid proposal prevails when two coordinators race.

Proposition 9.1. *Assume the head of job J is updated only by atomic compare-and-swap, and that every successful swap strictly increases the epoch. If two coordinators submit commits against the same (t, h), at most one receives a receipt, and the other receives* $\mathrm{StaleParent}(J, t, h)$.

Proof. The first swap to execute finds the head equal to (t, h) and replaces it with a pair of epoch $t + 1$. The second finds a head whose epoch differs from t, so its comparison fails and it returns the stale outcome. □

The stale-parent outcome is a supersession in the sense of Chapter 11. The attempt was valid, another valid attempt committed first, and the outcome says nothing against the gluing of the proposals, the invariant, or the fabric.

In an implementation the evidence may be a checked manifest holding interface digests and invariant witnesses. The strength of the resulting guarantee depends on what the verifier actually checks, which may range from digest equality, through recomputation of a sample, to full re-execution.

Requirement 9.1 (Atomic head update)**.** *Commit shall be implemented as an atomic compare-and-swap of the job head against the parent epoch and digest named in the proposal. A failed swap shall return the current head to the caller and shall be recorded as a supersession under Requirement 11.1.*

Remark 9.1. The word stale is used in two senses and they should not be confused. A stale parent means that the head moved between the time an attempt named it and the time its swap executed, and it is decided at the compare-and-swap. A stale proposal, in the sense of Chapter 12, means that the shard a module read has drifted from the current shard, and it is decided at admission against the staleness contract. A proposal can be stale in one sense and not the other.

Requirement 9.2 (Verifier statement)**.** *For each class of invariant the service shall publish what its verifier checks. Each receipt shall name the verification level applied, so that a customer can distinguish a digest comparison from a recomputation.*

Requirement 9.3 (Receipt content)**.** *A receipt shall name the job, the new epoch, the parent digest, and the digest of the evidence manifest, and shall reference the committed checkpoint of Chapter 6 that it authorizes. Receipts shall form a chain through their parent digests.*

Chapter 10

Feasibility of Optical Exchange

A commit is useful only if the boundary data it needs arrive before the commit is due. This chapter states the condition under which the fabric can deliver those data, and the placement rule that follows. The fabric carries the boundary data and nothing more. Whether the data glue, and whether the glued state meets the declared invariant, is checked at the destination, and a feasible schedule can carry proposals that fail either check.

Give each required exchange k a payload of b_k bits, a path P_k through the fabric, a release time r_k at which the sending module produces the data, and a delivery deadline d_k at which a commit needs it. For a fluid approximation allocate a rate $x_k(\tau) \geq 0$ on the reserved path. Let $C_e(\tau)$ be the usable capacity of link e at time τ, and let ℓ_k be the end-to-end latency allowance of exchange k. The schedule is feasible when

$$\forall e, \tau : \sum_{k:\, e \in P_k} x_k(\tau) \leq C_e(\tau), \qquad \forall k : \int_{r_k}^{d_k - \ell_k} x_k(\tau)\, d\tau \;\geq\; b_k,$$

with $x_k = 0$ outside the transmission window $[r_k, d_k - \ell_k]$.

Proposition 10.1. *Consider a single exchange with constant reservable path capacity B, release time 0, deadline $m\tau_i$, and latency λ. The fluid condition holds if and only if $b/B + \lambda \leq m\tau_i$, which is the negation of the refusal condition of Requirement 5.4.*

Proof. The best rate is the constant B, since any feasible allocation is bounded by it. The integral then equals $B\,(m\tau_i - \lambda)$, and the requirement that it be at least b rearranges to the stated inequality. □

Thus Requirement 5.4 is the single-exchange screening test, and the fluid model is its generalization to many exchanges that contend for shared links. After discretizing time into slots, the unknowns $x_{k,s} \geq 0$ appear linearly in both families of constraints, so feasibility of a given placement is decided by a linear program. The model is a first approximation for the scheduler. A packet-level deployment should validate its deadline assumptions against measured queueing and tail latency.

Requirement 10.1 (Reservable capacity)**.** *The capacity C_e used in feasibility tests shall be the service that the job can reserve on link e, which may be materially below the line rate of the optics. The service shall report the reservable figure and its basis.*

Requirement 10.2 (Placement by feasibility)**.** *A placement shall be accepted only if the fluid feasibility conditions hold for every exchange required by the declared contracts, with deadlines taken from the commit times those contracts imply. A replacement placement after failure shall be subject to the same test.*

10.1 A diagnostic for contended links

When an exchange fails the fluid test, the service should say why in terms that a customer can act on. For a single link the reason takes a simple form. Let K_e be the set of exchanges whose paths contain link e, and for an interval $[a, b]$ let $K_e(a, b)$ be those whose transmission windows $[r_k, d_k - \ell_k]$ lie inside $[a, b]$.

Proposition 10.2. *Work under the idealization of the fluid model: each payload is divisible and may be sent preemptibly at any rate the link allows, all of the payload of exchange k is available at its release time r_k, and the effective window of exchange k is $[r_k, d_k - \ell_k]$. If the fluid schedule is feasible then for every link e and every interval $[a, b]$,*

$$\sum_{k \in K_e(a,b)} b_k \;\le\; \int_a^b C_e(\tau)\, d\tau.$$

When all exchanges share a single link and the condition holds for every interval, defined with the effective windows, the schedule is feasible.

Proof. Each exchange in $K_e(a, b)$ transmits only inside its window, hence inside $[a, b]$, and must deliver b_k there. The total rate on e never exceeds C_e, so the total delivered in $[a, b]$ cannot exceed the integral. For sufficiency on one link, schedule the exchanges by earliest deadline among those released. A shortfall at the first missed deadline d exhibits an interval ending at d whose demand exceeds capacity, by the standard argument for preemptive deadline scheduling. □

With several links the condition remains necessary and is no longer sufficient, because an exchange that crosses two links must receive the same rate on both. A violated interval nevertheless identifies the binding link, the window, and the amount by which demand exceeds capacity, which is the evidence that the schedule class of Requirement 11.1 asks the rejection record to carry. Under the assumptions above, a violated interval is a witness of infeasibility that a customer can check by arithmetic, without trusting the scheduler: the link, the interval, the exchanges inside it, and the two sums.

The contended attempt of Chapter 11 shows the form. The uplink u_1 carries the two exchanges of module 1 and the competing transfer, with windows $[0, 0.48]$ s, so the demand over that interval is $60 + 60 + 40 = 160$ Gb against a capacity of $300 \times 0.48 = 144$ Gb. The binding link is u_1, the interval is $[0, 0.48]$, and the shortfall is 16 Gb. The remedies of Table 11.1 follow from the same inequality. Raising the capacity to $160/0.48 \approx 333.4$ Gb/s, widening the window by moving the deadline to $0.02 + 160/300 \approx 0.553$ s, or removing 16 Gb of demand from the interval each restores the condition, and each can be stated to the customer without describing the fluid model. The witness also separates the three choices open to the customer, which are to reduce the payload in the interval, to extend the effective deadline, and to reserve more capacity, and it shows how much of each is needed.

Together with Chapters 8 and 9 this yields the placement rule of the specification: place modules wherever their boundary exchanges can arrive on time, and commit only when the state assembled from the exchanged data is admissible. Chapter 14 turns the rule into an experiment.

Chapter 11

A Worked Example in Three Attempts

This chapter runs three commit attempts through the machinery of Chapters 8 to 10. Each is a candidate transition from the same head $(7, h_7)$ of a job J, proposed by three GPU modules that share a parameter block. The first fails to glue, the second glues but violates the power budget, and the third meets both conditions and is then tested against the fabric. The three failures differ in kind and call for different remedies, and only the last can be helped by more link capacity.

Definition 11.1 (Commit attempt)**.** *A commit attempt is a submission of module proposals against a named head (t, h). It is evaluated in three steps. First its exchanges are tested for feasibility, at admission with declared release times and again whenever an observed release time shortens a window, because the data against which gluing and the invariant are checked must arrive before they can be checked. Then the proposals are tested for gluing, and then the glued state is tested against the declared invariant. If it passes all three it is presented to the compare-and-swap of Chapter 9. A rejected attempt leaves the head unchanged. A retry is an attempt made after an earlier rejection, against the same head or a later one. Committed progress is the number of receipts, since only a receipt advances the logical epoch.*

Every attempt ends in exactly one outcome: a receipt, a rejection of one of the three classes of Requirement 11.1, or supersession. An attempt is superseded when it passes evaluation but loses the compare-and-swap and receives the stale-parent outcome of Chapter 9. Evaluation is performed against the head the attempt names, so the class of a rejection does not depend on whether another attempt has since advanced the head. A superseded attempt is not a rejection of any class. It records that the proposal was valid and that another valid attempt committed first.

In attempts A and B the fabric is uncontended and the fluid test passes, so both are evaluated for gluing and for the invariant. Attempts A and B are rejected and attempt C is eventually committed. All three start from the head $(7, h_7)$, which is the parent of epoch 8, and only the committed attempt advances the epoch.

11.1 Setup

The shared block has three coordinates (x, y, z), each a shard of the block. Every module holds a proposal $p_i \in \mathbb{R}^3$ and no private state, so each restriction to an interface is the identity and each pairwise interface carries $I_{ij} = \mathbb{R}^3$. The partition complex has three

module cells and three pairwise interface cells. No triple cell is needed for the block itself, because agreement of all three pairs implies agreement of all three copies.

Each module may change only a specified subspace of its proposal. The authorized repair spaces are

$$A_1 = \operatorname{span}\{e_x, e_y\}, \qquad A_2 = \operatorname{span}\{e_y, e_z\}, \qquad A_3 = \operatorname{span}\{e_z\},$$

with the maps $L_{i,ij}$ the inclusions into $\mathbb{R}^3$. The repair operator is

$$Da = (a_1 - a_2,\ a_1 - a_3,\ a_2 - a_3) \quad \text{on interfaces 12, 13, 23.}$$

Module 1 cannot touch z, module 2 cannot touch x, and module 3 can touch only z. Coordinate x can therefore be moved only by module 1, coordinate y by modules 1 and 2, and coordinate z by modules 2 and 3.

Proposition 11.1. *Given proposals p_1, p_2, p_3, a compatible family reachable by authorized repairs exists if and only if $p_{2,x} = p_{3,x}$. When it exists the glued state is unique and equals $q = (p_{2,x},\ p_{3,y},\ p_{1,z})$.*

Proof. A common value q must satisfy $q - p_i \in A_i$ for each i. In coordinate x modules 2 and 3 cannot move, so q_x equals both $p_{2,x}$ and $p_{3,x}$, which forces them to be equal, and module 1 can move to meet them. In coordinate y module 3 cannot move, so $q_y = p_{3,y}$, and modules 1 and 2 can move to meet it. In coordinate z module 1 cannot move, so $q_z = p_{1,z}$, and modules 2 and 3 can move to meet it. Conversely these choices are reachable by authorized repairs. □

The same fact appears in the language of Proposition 8.1. The operator D is injective, since $A_1 \cap A_2 \cap A_3 = \{0\}$, so $\operatorname{im} D$ has dimension 5. Defects that arise from proposals are cocycles, satisfying $\delta_{12} - \delta_{13} + \delta_{23} = 0$, and the cocycles form a space of dimension 6. The functional $\varphi(\delta) = (\delta_{23})_x$ vanishes on $\operatorname{im} D$ because a_2 and a_3 have no x component, and it is not zero on the cocycles, so it cuts out $\operatorname{im} D$ exactly. For defects arising from proposals, $[\delta] = 0$ if and only if $\varphi(\delta) = p_{2,x} - p_{3,x} = 0$.

The global invariant is a power budget on the glued block. For illustration take the linear model

$$P(q) = 4q_x + 3q_y + 2q_z \text{ kW}, \qquad \mathrm{Inv}(q):\ P(q) \le 40.$$

The exchange fabric attaches each module to an optical Ethernet switch through a 400 Gb/s port, of which 300 Gb/s per direction is reservable under the operator's policy. The latency allowance for each exchange is $\ell = 0.02$ s. The reservable fraction is an assumed policy value and is not a figure from any vendor.

11.2 Attempt A: a defect outside the image of the repair operator

The modules propose

$$p_1 = (2, 1, 3), \qquad p_2 = (4, 1, 0), \qquad p_3 = (1, 3, 0).$$

The interface defects are $\delta_{12} = (-2, 0, 3)$, $\delta_{13} = (1, -2, 3)$, and $\delta_{23} = (3, -2, 0)$, which satisfy the cocycle identity. Here $\varphi(\delta) = 3 \neq 0$, so $\delta \notin \operatorname{im} D$. In the rank formulation, $\operatorname{rank} D = 5$ while the augmented matrix $[\, D \mid \delta \,]$ has rank 6.

What averaging does. Ordinary parameter averaging produces $\bar{q} = (7/3,\ 5/3,\ 1)$, and every module adopts it, so the copies agree everywhere. The changes this demands are

$$\bar{q} - p_1 = (\tfrac{1}{3}, \tfrac{2}{3}, -2), \qquad \bar{q} - p_2 = (-\tfrac{5}{3}, \tfrac{2}{3}, 1), \qquad \bar{q} - p_3 = (\tfrac{4}{3}, -\tfrac{4}{3}, 1).$$

Module 1 is asked to change z by -2 without authority over z. Module 2 is asked to change x by $-5/3$ without authority over x. Module 3 is asked to change x and y without authority over either. The average agrees, and it requires every module to act outside its authorization.

What the repair operator adds. The operator separates what can be agreed from what cannot. Coordinates y and z can be agreed at $q_y = 3$ and $q_z = 3$: module 1 raises y by 2, module 2 raises y by 2 and z by 3, and module 3 raises z by 3, each within its authorized subspace. The coordinate x cannot be agreed. Modules 2 and 3 hold 4 and 1 and neither may move it, and module 1 can match only one of them. The obstruction is the x component of interface 23, with value 3.

Remedy. The remedy lies in what the modules are authorized to repair. If module 3 is also authorized to change x, so that $A_3' = \operatorname{span}\{e_x, e_z\}$, the pinning argument gives the unique state $q = (4, 3, 3)$, with $P(q) = 31 \le 40$. Alternatively module 2 or module 3 can re-propose a different x. No amount of link capacity changes $\operatorname{im} D$ (Proposition 8.2).

11.3 Attempt B: a glued state that exceeds the budget

The modules propose

$$p_1 = (3, 2, 5), \qquad p_2 = (4, 2, 1), \qquad p_3 = (4, 6, 1).$$

Now $p_{2,x} = p_{3,x} = 4$, so the proposals glue, and the glued state is forced to be $q = (4, 6, 5)$. The repairs are $a_1 = (1, 4, 0) \in A_1$, $a_2 = (0, 4, 4) \in A_2$, and $a_3 = (0, 0, 4) \in A_3$, and each $p_i + a_i = q$. The power is

$$P(q) = 4 \cdot 4 + 3 \cdot 6 + 2 \cdot 5 = 44 \text{ kW},$$

which exceeds the budget by 4 kW. Because Proposition 11.1 makes the glued state unique, the repair operator offers no alternative compatible state to choose from, so the invariant cannot be met by repair. The remedy is a different globally admissible state, obtained by changing the proposals that pin the state, or a different placement with different authorizations. The pinned coordinates here are y, set by module 3, and z, set by module 1. Lowering $p_{3,y}$ from 6 to 5 alone gives $P = 41$, still over budget, and Attempt C also lowers $p_{1,z}$ from 5 to 4.

11.4 Attempt C: a state that glues and meets the budget

The modules propose

$$p_1 = (3, 2, 4), \qquad p_2 = (4, 2, 1), \qquad p_3 = (4, 5, 1).$$

Again $p_{2,x} = p_{3,x}$, and the glued state is $q = (4, 5, 4)$. The repairs are $a_1 = (1, 3, 0) \in A_1$, $a_2 = (0, 3, 3) \in A_2$, and $a_3 = (0, 0, 3) \in A_3$. Each satisfies $p_i + a_i = (4, 5, 4)$, and

$$P(q) = 16 + 15 + 8 = 39 \text{ kW} \le 40.$$

The attempt is admissible as a state. Whether it can be committed depends on whether the boundary data arrive in time.

Exchanges and deadlines. Each module receiving a neighbour's block computes the interface defect against its own proposal. Three exchanges are required: module 1 sends its block to module 2, module 1 sends it to module 3, and module 2 sends its block to module 3. Each payload is the full block of $b = 60$ Gb (7.5 GB). Write u_i for the uplink of module i and d_i for its downlink. The paths are $P_1 = \{u_1, d_2\}$, $P_2 = \{u_1, d_3\}$, and $P_3 = \{u_2, d_3\}$. All blocks are ready at $r_k = 0$, and the commit needs the data by $d_k = 0.5$ s, so the transmission window is [0, 0.48] s.

Each exchange passes the single-exchange screening test of Requirement 5.4, since $60/300 + 0.02 = 0.22 \leq 0.5$. The fluid condition accepts the schedule that sends every exchange at a constant 125 Gb/s, which delivers $125 \times 0.48 = 60$ Gb. The loads are $x_1 + x_2 = 250$ Gb/s on u_1, $x_2 + x_3 = 250$ Gb/s on d_3, and 125 Gb/s on each of u_2 and d_2, all within the 300 Gb/s reservable capacity.

A competing transfer. Suppose a transfer of 40 Gb leaves module 1 over the same uplink in the same window, for instance a checkpoint upload belonging to another job. Taken alone it passes the screening test, since $40/300 + 0.02 \approx 0.15 \leq 0.5$. The demand on u_1 in the window becomes $60 + 60 + 40 = 160$ Gb, while the link can carry at most $300 \times 0.48 = 144$ Gb. The schedule is infeasible by 16 Gb, although every individual exchange passed its screening test. The attempt is rejected on schedule alone, and its record marks gluing and the invariant as not evaluated, although the proposals would pass both.

This failure responds to capacity and to schedule. It disappears if the reservable capacity on u_1 is at least $160/0.48 \approx 333.4$ Gb/s, which is within the 400 Gb/s line rate of the port. It also disappears if the deadline moves to $0.02 + 160/300 \approx 0.553$ s, or if the competing transfer is moved outside the window. It does not depend on the proposals, and no change to authorizations or to the power budget affects it.

Commit. Remove the competing transfer. The three exchanges complete within their window, and the control plane assembles an inhabitant of $\mathrm{Ready}(7, h_7)$ consisting of the three proposals, the compatibility evidence, and the invariant evidence. The compatibility evidence is the repair triple (a_1, a_2, a_3) together with the checks $a_i \in A_i$ and $p_i + a_i = q$. The invariant evidence is the computation $P(q) = 39 \leq 40$. The verifier recomputes both from the proposals, so the receipt records full recomputation as its verification level (Requirement 9.2).

The compare-and-swap of the head of J against $(7, h_7)$ succeeds and returns $\mathrm{Receipt}(J, 8)$. The receipt names the parent digest h_7, the digest m_8 of the evidence manifest, and the new head $(8, h_8)$, where h_8 is the digest of the committed checkpoint it authorizes. A second coordinator that submits the same evidence against $(7, h_7)$ after the swap receives $\mathrm{StaleParent}(J, 7, h_7)$ together with the current head $(8, h_8)$, as Proposition 9.1 requires.

11.5 Summary of the attempts

Attempt	Condition that fails	Evidence	Remedy
A	Gluing	$\varphi(\delta) = 3$, defect outside $\operatorname{im} D$	Enlarge authorized repairs, or change the proposals; capacity does not help
B	Invariant	$P(q) = 44 > 40$ with a unique glued state	Different proposals, or a different placement; capacity does not help
C with contention	Schedule	160 Gb demand against 144 Gb capacity on u_1	Raise reservable capacity to about 334 Gb/s, relax the deadline, or move the competing transfer
C without contention	None	Receipt for epoch 8	Commit

Table 11.1: The outcomes of the attempts in the worked example, with the remedy appropriate to each failure.

11.6 Counting attempts

Counting separates activity from progress. Take attempt C twice: first with the competing transfer present, where it is rejected for its schedule, and then resubmitted against the same head with the contention removed. The tally is four attempts (A, B, the contended C, and the resubmitted C), three rejections, one in each class, one retry, and one receipt. Committed progress is a single epoch, from 7 to 8. Write n_a for the number of attempts, n_r for receipts, n_d, n_i, n_s for rejections of the defect, invariant, and schedule classes, and n_x for superseded attempts. Every attempt has exactly one outcome, so

$$n_a \;=\; n_r + n_d + n_i + n_s + n_x.$$

The commit yield per attempt is $\kappa = n_r/n_a$, and the decided yield is $\kappa_{\mathrm{dec}} = n_r/(n_a - n_x)$, which leaves out superseded attempts. Here $n_x = 0$ and both equal $1/4$. Suppose now that a redundant coordinator submits the same evidence for the resubmitted attempt C and loses the swap. Then $n_a = 5$, $n_x = 1$, $\kappa = 1/5$, and $\kappa_{\mathrm{dec}} = 1/4$. The redundancy lowers κ and leaves κ_{dec} unchanged, and neither the fabric nor the repair mechanism misbehaved. A third quantity, the committed epochs per unit time θ, is measured over wall-clock time and is reported with both yields, because an aggressive retry policy can lower κ while raising θ.

Requirement 11.1 (Rejection record)**.** *Every rejected attempt shall produce a record naming the parent head, the class of the rejection, the evidence for that class, and the remedy it points to. The classes are defect, meaning an incompatibility on an overlap that no authorized repair removes; invariant, meaning a glued state that violates a declared global invariant; and schedule, meaning that no feasible allocation delivers the required boundary data by its deadline. An attempt whose exchanges are predicted to be infeasible shall be rejected on schedule alone, and its record shall mark gluing and the invariant as not evaluated. Otherwise gluing and the invariant are both evaluated, and the record shall list each class that failed. An attempt that passes evaluation and loses the compare-and-swap shall*

instead produce a supersession record naming the parent head, the current head returned by the swap, and the attempt that committed. A supersession shall not be counted in any rejection class. The service shall report for each job the tallies n_a, n_r, n_d, n_i, n_s, n_x and the number of retries, and the reconciliation $n_a = n_r + n_d + n_i + n_s + n_x$ shall hold.

The Part III chapters reuse this machinery with application-specific invariants. In training the shared block is a parameter shard with a staleness contract, and in transcription the shared state is the set of mention records with their audio spans, whose invariant is the unresolved-status rule of Chapter 7. Each chapter gives a locally valid result that the service withholds or revises, and shows how the rejection record tells the customer which class occurred.

Part III

Applications

Chapter 12

Hierarchical Synchronization and Bounded Staleness

In synchronous data-parallel training every worker exchanges gradients at each step. In a modular cloud the cost of that exchange differs sharply between workers in the same module and workers in different modules. The hierarchical scheme places frequent exchange inside each module and exchanges between modules at a longer declared interval, which is the parameter of Requirement 5.3. The logical epoch of the job advances only when an inter-module commit succeeds, so the steps taken inside a module are local transitions in the sense of Chapter 2 and the committed epochs are the global progress.

The comparison to be made is between full synchronization across all modules at every step and periodic exchange between modules, at a matched validation target, with wall-clock time and energy to reach it as the reported quantities. A result that reaches the target faster with a longer interval has not shown the interval to be free. It has shown that for this model and data the contract was sufficient, and sufficiency is an empirical finding about the workload. This chapter fixes what the contract says about stale contributions and shows how a customer can tell why a contribution was withheld.

12.1 The shared shard and its history

Let w_s denote the shared parameter shard at committed version s. Each receipt records the norm of the update it committed, $u_s = \|w_{s+1} - w_s\|$ under a norm declared by the job. A proposal from module i is a triple (v, d_v, g_i) consisting of the version v of the shard it read, the digest d_v of the shard it actually read, and its gradient or accumulated update $g_i = g_i(w_v)$. When the proposal is presented against head (t, h) with $t \geq v$, two quantities describe how far it has aged. The version lag is $\tau = t - v$. The certified drift is

$$\hat{\Delta}_{v,t} \;=\; \sum_{s=v}^{t-1} u_s.$$

Proposition 12.1. *For any versions $v \leq t$, $\|w_t - w_v\| \leq \hat{\Delta}_{v,t}$. If g_i is L_i-Lipschitz on a set containing w_v and w_t, then*

$$\|g_i(w_v) - g_i(w_t)\| \;\leq\; L_i \|w_v - w_t\| \;\leq\; L_i\, \hat{\Delta}_{v,t}.$$

Proof. The first inequality is the triangle inequality applied to the telescoping sum $w_t - w_v = \sum_{s=v}^{t-1} (w_{s+1} - w_s)$. The second is the Lipschitz hypothesis, and the third follows from the first. □

History	Lag	Norms since base	$\hat{\Delta}$	$L_i\hat{\Delta}$	Lag only	Contract
First	1	0.60	0.60	1.20	admit	reject
Second	4	$0.03, 0.04, 0.02, 0.03$	0.12	0.24	reject	admit

Table 12.1: Two histories ending at version 40, with $L_i = 2.0$, $\varepsilon_i = 0.5$, and $\tau_{\max} = 5$. The lag-only rule admits when the lag is at most 2. The values are illustrative.

The claim is scoped to the discrepancy between two gradient evaluations: the one the worker computed at w_v and the one it would have computed at the current shard w_t. It says nothing about convergence, and training quality remains something the prototype measures at a matched validation target. The certified drift can be computed from receipts alone, so the service does not need to retain old versions of the shard. When every committed update has norm at most δ the sum is at most $\tau\delta$, which recovers the uniform bound of Proposition 2.3; the sum is the sharper statement because it uses the updates that actually occurred.

Definition 12.1 (Staleness contract)**.** *A staleness contract for a shard consists of a lag limit $\tau_{\max}$, and for each module i a Lipschitz constant L_i with its declared basis and a tolerance ε_i on gradient discrepancy. A proposal read at version v is admitted unchanged against head t when*

$$t - v \le \tau_{\max} \qquad \text{and} \qquad L_i\,\hat{\Delta}_{v,t} \le \varepsilon_i.$$

Both conditions are needed. A one-version lag can fail after a large update, and a longer lag can pass after several small ones. Take $L_i = 2.0$, $\varepsilon_i = 0.5$, and $\tau_{\max} = 5$, which are illustrative values. Table 12.1 compares two histories that end at version 40. In the first, the last update had norm 0.60 and the proposal was read one version earlier. In the second, the proposal was read four versions earlier and the four intervening updates were small. A rule on lag alone with limit 2 would admit the first proposal, whose gradient discrepancy bound is 1.20, more than twice the tolerance, and would reject the second, whose bound is 0.24. The contract above reverses both decisions.

Requirement 12.1 (Base digest)**.** *Every inter-module proposal shall carry the version and the digest of the shard it read. A proposal whose digest matches no committed version shall be rejected as a defect.*

Requirement 12.2 (Certified drift)**.** *Receipts shall record the norm of each committed update under the declared norm, so that certified drift can be computed by summation without retaining earlier shard versions.*

Requirement 12.3 (Two-part admission)**.** *The service shall test both the version lag and the certified drift against the staleness contract. The Lipschitz constant of each module shall be declared by the application together with its basis, measured or assumed, and the service shall report the product $L_i\hat{\Delta}$ with every rejection it makes on this ground.*

A proposal that fails either test is a defect on the interface between the proposal and the head. The repair is a rebase, and the defect lies in the image of the repair operator exactly when the module's authorized repairs include a rebase.

12.2 A locally valid gradient that the service withholds or revises

Take the first history of the previous section. The head is $(40, h_{40})$, modules 1 and 2 present proposals read at version 40 with lag 0 and no drift, and module 3 presents a proposal read

at version 39. Its gradient $g_3(w_{39})$ is correct for the shard it read, and nothing is wrong with it locally. It fails admission because $L_3\hat{\Delta}_{39,40} = 1.20 > 0.5$. Three commit attempts from the same head show how the failure resolves depending on what else is true of the module and of the fabric. In each attempt the only difference lies in module 3.

Two operations must be kept apart. A stale application admits a proposal computed at w_v and uses it unchanged against the head w_t, which is permitted only when the staleness contract holds. A rebase discards the proposal and has the module read the current shard and recompute its gradient at that shard on the same data. A rebase is not the application of an old gradient to a new shard. It produces a new proposal $g_i(w_t)$ with its own base digest, and the receipt records which of the two operations occurred. A rebase costs a read of the current shard, the recomputation, and the transfer of the new proposal, and all three are charged to the deadline of the attempt. If the head advances while a rebase is in progress, the rebased proposal still names the parent whose shard the worker actually read. It is evaluated against that parent, where its lag and drift are zero, and if it passes all three tests it reaches the compare-and-swap, finds a different head, and is superseded. It is not rejected, because nothing about it failed evaluation. The module then submits a new attempt against the new head, which is a retry, and the lag and drift of that attempt are computed against the new head, using the update norm recorded on the receipt of the competing commit.

Attempt S1: withheld, a defect outside the repair class. Module 3 reads from a data stream whose microbatch is consumed and discarded, so it has no authority to recompute. Its authorized repairs contain no rebase, and the drift defect lies outside the image of the repair operator. The attempt is rejected with class defect and the evidence $L_3\hat{\Delta} = 1.20 > 0.5$ with no authorized rebase. The remedies concern authorization and contract: authorize the module to buffer its microbatch so that a rebase becomes possible, raise ε_3 to at least 1.20, or adopt a declared contract in which a commit may proceed without a module that fails admission, with the exclusion recorded. More link capacity changes none of these.

Attempt S2: revised, a defect inside the repair class. Module 3 may rebase. The shard is 40 Gb, so at 300 Gb/s of reservable capacity the read takes $40/300 \approx 0.133$ s. The recomputation takes 0.25 s. The new proposal is also 40 Gb, and sending it takes another 0.133 s. Each of the two transfers carries a latency allowance of 0.02 s. The total charged to the deadline is about 0.557 s, and with a deadline of 0.65 s the repair fits. The service revises the contribution: module 3's proposal is replaced by $g_3(w_{40})$, which has lag 0 and no drift, the attempt passes admission, and it commits. The receipt for epoch 41 records a rebase, the version from which module 3 was rebased, and the digest of the discarded base.

Attempt S3: withheld, a schedule problem. Everything is as in S2 except that the deadline is 0.5 s. The rebase cannot complete in time, and the attempt is rejected with class schedule, short by about 0.057 s. Here capacity can help. The condition $0.25{+}0.04{+}80/C \leq 0.5$ requires $C \geq 80/0.21 \approx 381$ Gb/s, which is below the 400 Gb/s line rate, so a larger reservation would admit the rebase. Relaxing the deadline to 0.557 s would do the same.

Counting follows Chapter 11. There are three attempts against head $(40, h_{40})$. One yields a receipt, so committed progress is a single epoch, from 40 to 41. Two are rejected, one for a defect and one for a schedule, none is superseded, and the commit yield is $\kappa = \kappa_{\mathrm{dec}} = 1/3$. The rejections are recorded before any compare-and-swap, so they leave the head unchanged. A customer reading the two rejection records learns different things.

The record for S1 points to authorization and contract, and the record for S3 points to the fabric and the deadline.

Requirement 12.4 (Admissibility of rebase)**.** *Whether a module may rebase shall be part of its declared authorized repairs. A rebase performed by the service shall be recorded in the receipt as a rebase, with the original version and digest, and never as a stale application. The recomputation and both transfers shall be charged to the deadline of the attempt.*

The remaining class, an invariant failure, arises in training when the glued update violates a declared global condition such as a bound on the norm of the combined step. It is handled as in Chapter 11: the glued state is checked against the invariant, a violation is rejected with class invariant, and the remedy is a different admissible state or placement.

Chapter 13

Context-Aware Assembly and Unresolved Names

Consider an audio transcription job in which GPU workers transcribe overlapping chunks in parallel. Each chunk result is locally valid, in the sense that it is a plausible transcription of its audio. Global admissibility is a property of the assembled transcript, in which a name in one segment may be constrained by context in another. The recognition pass is expensive and runs concurrently. The context needed to check the result is comparatively small and crosses module boundaries afterward. This chapter makes the unresolved-name rule of Chapter 7 an invariant of the delivered transcript and shows how a customer can tell a defect, an invariant violation, and a schedule failure apart.

13.1 Mentions, evidence, and the transcript invariant

Definition 13.1 (Mention record)**.** *A mention record for a name in the audio consists of its audio span σ_m, its first-pass rendering r_m, and a status that is either canonical, carrying a spelling c_m and the evidence E_m for it, or unresolved, carrying a set C_m of candidate spellings that contains r_m.*

The kinds of evidence differ in strength, and the job declares which kinds suffice. Three are distinguished here. Direct corroboration is a source named in the job record that spells the name, such as a customer-supplied glossary, on-screen text, or the title and description of the recording. A resolved role clue is a role or title phrase in the audio that a named reference source maps to a spelling. A relational aside is a phrase indicating how one name relates to another, such as a remark that two people share a surname, without fixing any spelling.

Definition 13.2 (Evidence standard and transcript invariant)**.** *The evidence standard requires that a canonical spelling differing from the first-pass rendering be supported by direct corroboration or by a resolved role clue, from a source named in the record. A relational aside may narrow a candidate set and may prompt comparison of two names, but it shall not select among candidates. The transcript invariant* Inv_T *holds when every mention carries its span and first-pass rendering, every canonical status meets the evidence standard, and every unresolved status carries a candidate set that contains the first-pass rendering and every candidate considered.*

The invariant is the conjunction of Requirements 7.1 to 7.3 over all mentions, and it makes no demand that every mention be resolved. A transcript with unresolved mentions satisfies Inv_T provided each unresolved mention is recorded as such.

13.2 Overlaps and the discrete repair class

Chunks of sixty seconds with ten seconds of overlap, numbered from zero so that chunk n spans seconds $50n$ to $50n+60$, give the partition complex a chain of pairwise interface cells, one for each overlap. The interface data on an overlap are the mention records that fall inside it. Two chunk workers may render the same mention differently, which is a defect on the overlap. The repair operator of Chapter 8 is linear, and here the repairs are discrete, and which repairs are available is a matter of the job's repair policy. Selection replaces a worker's rendering by another entry of its own n-best list and creates no new candidate. Regeneration re-decodes the chunk, for example with a wider context or a glossary bias, and may produce candidates that were absent from the original list. It is a separate operation with a GPU cost, a provenance record, and a charge against the deadline. The repair policy names which of the two the job authorizes.

Proposition 13.1. *Under a selection-only repair policy, the defect on the overlap of two chunks is repairable if and only if their n-best lists for the mention have a common entry. Under a policy that also authorizes regeneration, disjoint lists do not by themselves make the defect unrepairable, and whether it is repaired depends on the lists that regeneration produces.*

Proof. A repair makes the two renderings equal, which requires a rendering that belongs to both lists. Conversely any common entry may be chosen by both workers. □

The proposition is the discrete counterpart of Proposition 8.1. A defect whose lists are disjoint lies outside the selection repair class, and no change to the fabric alters that. It moves inside the repair class only if the authorized repairs are enlarged to include regeneration, which is a change of policy and not of capacity.

13.3 A worked case with four commit attempts

The recording is forty minutes of narration. A first mention M_1 occupies 12:30.8 to 12:32.9, which is 750.8 to 752.9 seconds and lies in the overlap of chunks 14 and 15. Shortly afterward, at 12:36, the speaker says that the person works as the chief executive of Nvidia. A second mention M_2 occupies 31:42 to 31:44 and lies in the overlap of chunks 37 and 38. The first pass renders it as Dana Wang, a fictional name used only for this controlled test. At 31:44.5 the speaker adds that the two people are, of course, no relation. Four commit attempts are made against the transcript job's head $(3, h_3)$.

Attempt W1: withheld, a defect outside the repair class. Chunk 14 renders M_1 as Jensen Wong with n-best list $\{\text{Jensen Wong}, \text{Jensen Wang}\}$, and chunk 15 renders it as Jensen Hwang with list $\{\text{Jensen Hwang}, \text{Jensen Huang}\}$. Each rendering is a plausible reading of its chunk. The lists are disjoint, so under the selection-only policy of this job the defect cannot be repaired by Proposition 13.1. The attempt is rejected with class defect. The remedy is to authorize regeneration, for instance a glossary-biased re-decode of chunk 14. The re-decode is a separate operation, recorded with its provenance, and its GPU time and transfers are charged to the deadline. Whether it produces Jensen Huang is something to be observed and is not assumed.

Attempt W2: withheld, an invariant violation with an admissible alternative. The chunk lists now glue. The assembly stage proposes to change the first-pass surname of M_2 to Huang, citing the aside that the two people are no relation. Every chunk result is

locally valid and the overlaps agree, but the proposed canonical status rests on a relational aside alone, which the evidence standard does not accept. The attempt is rejected with class invariant. The different admissible state is available here, unlike in Chapter 11: the same transcript with M_2 unresolved and C_{M_2} containing both Wang and Huang satisfies Inv_T.

Attempt W3: withheld, a schedule failure that capacity cannot help. The assembler needs the mention records of chunk 38 by a deadline of 2.0 s after the assembly pass opens. The worker for chunk 38 is a straggler that finishes decoding at 1.995 s. With latency allowance $\ell = 0.02$ s the transmission window is [1.995, 1.98], which is empty. The fluid condition of Chapter 10 fails whatever the capacity, because $d_k - \ell_k - r_k = -0.015 < 0$. The remedies are a later deadline or an earlier release, for example by rebalancing the work, and a faster link has no effect. The payloads are small, about 0.2 Gb per chunk, which at 300 Gb/s takes under a millisecond, so contention for bandwidth is not the cause. The window is empty only once the observed release time is known, so the schedule test is applied again at that moment, and the attempt is rejected on schedule alone with gluing and the invariant marked as not evaluated.

Attempt W4: committed with an unresolved mention. In this attempt the job authorizes regeneration, and a glossary-biased re-decode of chunk 14 is performed as the separate operation described in W1, with its provenance recorded and its GPU time and transfers charged to the deadline. The manifest glossary contains the entry Jensen Huang and supplies it as the bias, and in the controlled recording the re-decode returns a list that contains it. The two lists now share an entry, and selection repairs the defect on the overlap, with both workers selecting it. M_1 becomes canonical with spelling Jensen Huang. Its evidence is the resolved role clue of the chief executive of Nvidia at 12:36, together with the named reference source that maps the role to the spelling, and the record preserves both first-pass renderings and their spans. The second mention M_2 is committed as unresolved, with candidates Dana Wang and Dana Huang. The aside is retained as a relational clue that gives a reviewer a reason to compare the surnames of the two people, and it is not used to choose between the candidates. The transcript satisfies Inv_T, the mention records reach the assembler within their window, and the attempt receives a receipt for epoch 4 that reports one canonical correction and one unresolved mention, and records the regeneration as an operation of its own.

Counting follows Chapter 11. There are four attempts, one receipt, three rejections, one in each class, and no supersession, so $\kappa = \kappa_{\mathrm{dec}} = 1/4$ and committed progress is one epoch. The customer reading the three rejection records learns that W1 concerns what the chunk workers were authorized to repair, that W2 concerns the evidence standard, and that W3 concerns the deadline and the release time. The committed transcript shows that an unresolved name is a legitimate part of a successfully committed result and not a failure of the job.

Requirement 13.1 (Declared evidence standard)**.** *The job shall declare which kinds of evidence suffice for a canonical spelling and which named sources may supply them. A relational aside shall not be accepted as sufficient evidence for a spelling, and the service shall record when an aside was used to narrow a candidate set.*

Requirement 13.2 (Declared repair policy)**.** *The job shall declare whether its repair policy is selection only or also authorizes regeneration. Selection and regeneration shall be recorded as separate operations. A regeneration shall carry its provenance, the context or glossary supplied to it, and its cost, and its cost shall be charged to the deadline of the attempt.*

Requirement 13.3 (Unresolved status is committable)**.** *A transcript containing unresolved mentions shall be committable provided each unresolved mention satisfies the candidate-set condition of the transcript invariant. The receipt shall report the number of canonical corrections and the number of unresolved mentions.*

The receipt reports the counts of canonical corrections and of unresolved mentions. The prototype measures, against a reference transcript, the exact name accuracy, the number of incorrect corrections, and the rate of unresolved names.

13.4 The delivered record

The transcript committed at epoch 4 is delivered with its record. For M_1 the record shows the canonical spelling Jensen Huang, the two first-pass renderings and their spans, the evidence class (a resolved role clue), and the reference source that mapped the role to the spelling. For M_2 it shows the unresolved status, the candidates Dana Wang and Dana Huang, and the aside retained as a relational clue. Each of these items corresponds to a requirement of Chapter 7: the original candidates and spans are preserved, each correction carries evidence a customer can examine, and the unresolved item keeps all its candidates and is not resolved by a rule that is not recorded.

The record also says what would change the outcome. Suppose a reviewer later supplies direct corroboration for M_2, for example an entry in a glossary belonging to the manifest that names the second person. A new attempt against the head $(4, h_4)$ proposes M_2 as canonical with that evidence. The evidence standard is met, the transcript satisfies the invariant, and the attempt commits at epoch 5 with a receipt reporting two canonical corrections and no unresolved mention. The transcript of epoch 4 remains in the chain, unaltered, and shows what was known when it was committed. A reviewer who instead proposes a spelling without evidence, perhaps because the aside seems suggestive, meets the same invariant violation as attempt W2. In both cases the rejection or the receipt records what the evidence was and what it supported, and the customer can see that the unresolved name was held open by the evidence standard and was not dropped by the pipeline.

Part IV

Evaluation and Procurement

Chapter 14

Experimental Protocols

The prototype has two GPU modules, each a single server, an optical Ethernet switch between them, and a control plane with redundant coordinator replicas. Hardware is rented or borrowed, and link speed is chosen to fit the budget. This chapter turns the requirements of Part II and the examples of Part III into runs. Each run is specified in a fixed order: the initial checkpoint, what is varied, when the fault or contention is injected, what is recorded, and what counts as passing. Passing is defined on the correctness of records and of classification. A performance quantity is reported with its spread and enters a decision only through the rule of Chapter 15. The numerical values that appear in the model examples of Chapters 11 to 13 serve as the design of the corresponding runs and are replaced by the measured shard sizes, compute times, and reservable capacities of the actual apparatus before any run.

14.1 Common apparatus and accounting

Every run keeps the counters of Chapter 11: attempts n_a, receipts n_r, rejections n_d, n_i, n_s by class, supersessions n_x, and retries, with the reconciliation $n_a = n_r + n_d + n_i + n_s + n_x$. Three summary quantities are reported together for every run. The commit yield per attempt is $\kappa = n_r / n_a$. The decided yield is $\kappa_{\mathrm{dec}} = n_r / (n_a - n_x)$. The throughput θ is the number of committed epochs per unit of wall-clock time, measured from the first attempt of the run to its last receipt. A retry policy that resubmits at once after a rejection adds attempts and so lowers κ, while it may shorten the time to the next receipt and so raise θ. A policy that waits has the opposite effect. Neither number ranks retry policies alone, so any run that varies a retry policy reports all three. A retry that resubmits unchanged proposals is meaningful only after a schedule rejection, since a defect outside the repair image and an invariant violation recur unless the proposals or the authorizations change.

Redundant coordinators produce superseded attempts. These are counted in n_x and in no rejection class, κ_{dec} leaves them out, and every run that uses more than one coordinator reports n_x separately. This keeps redundancy from being mistaken for a fault of the fabric or of the repair mechanism, both of which may have behaved exactly as specified.

Every configuration is run at least three times with distinct seeds. All repeats are reported with their minimum and maximum, and no result is chosen from among them. Before the first run a manifest is written. It records the digest of the initial checkpoint, the software versions, the partition and placement, the reservation table, the staleness contract, the repair policy, the evidence standard, the retry policy, the injection schedule, and the values of every symbol used in a pass criterion, such as the validation target V^{*} and the outcome-class tolerances. These values are not changed after a run.

Three conditions are part of every pass criterion and are not repeated below. The counters reconcile. The receipt chain verifies from the initial checkpoint to the final head, each receipt naming the digest of its parent. Every rejection and supersession carries a record of the form required by Requirement 11.1.

14.2 Contract runs

Run C1, admission and contract checks. *Initial checkpoint.* None is required, since no job starts. The run uses job templates. *Varied.* Five submissions: a request stating only a device count; a job placed on part of a rack whose placement report is requested; a job that performs a cross-module transfer covered by no contract; a job with an interface that carries only a digest and declares nothing about summaries; and a job with a global power budget that is not declared as an invariant. *Injected.* Nothing. *Recorded.* The response of the service to each submission. *Passes when.* The device-count request is recorded with the default shape of Requirement 4.1. The placement report contains no committed figure for a scope larger than the one allocated. The uncovered transfer is refused or logged as unsanctioned. The digest-only interface is refused or flagged as a summary interface. The undeclared invariant is refused, or must be declared before the job is accepted.

14.3 Training runs

Run T1, interval comparison. *Initial checkpoint.* A model initialized from a fixed seed, at shard version 0 with digest d_0 and head $(0, h_0)$, the same for every configuration, with a seeded data order. *Varied.* The inter-module exchange interval $m \in \{1, m_1, m_2\}$, where $m = 1$ is full synchronization at every step and $m_1 < m_2$ are longer declared intervals. The staleness contract is fixed across configurations. *Injected.* Nothing. The reservable capacity is fixed and recorded. *Recorded.* Wall-clock time and energy to reach the validation target V^*, the quantities κ, κ_{dec}, and θ, the validation outcome of every exchange, the version lag and certified drift of every admitted proposal, and the exposed communication fraction φ defined in Chapter 15. *Passes when.* Each configuration either reaches V^* within the budget fixed in the manifest or is reported as not having reached it, and the common conditions hold. The comparison between intervals is reported at matched V^* and is not itself a pass condition.

Run T2, staleness, rebase, and withheld contributions. *Initial checkpoint.* A checkpoint at committed version 40 taken from a T1 run, whose receipts record the update norms of versions 36 to 40. For the controlled conditions the norms of chosen committed updates are set by scaling the step size for those steps, so that one update is large and the others small, with values from the manifest. Module 2 plays the stale module. *Varied.* The eight conditions of Table 14.1. *Injected.* The staleness is created by holding back module 2's proposal for a chosen number of versions. In condition (f), one competing commit is injected at half the elapsed rebase time. In condition (g) the digest of the shard read is altered before the proposal is sent. Condition (h) is created by holding module 2 back past the lag limit while the intervening updates stay small. *Recorded.* For each condition, the lag, the certified drift, the product $L_i\hat{\Delta}$, the measured discrepancy $\|g_i(w_v) - g_i(w_t)\|$ computed offline from the retained shards, and an estimate $\hat{L}_i$ of the Lipschitz constant by finite differences. The operation performed is recorded as a stale application or a rebase. For a rebase the times of the shard read, the recomputation, and the transfer of the new proposal are recorded, with their sum and the deadline. The class of any rejection is recorded. For

Case	Stale module and history	Deadline (s)	Capacity (Gb/s)	Expected outcome
(a)	Lag 4, four small updates	not binding	300	Stale application, receipt
(b)	Non-rebasable, lag 1, one large update	any	any	Rejected, defect
(c)	Rebasable, lag 1, one large update	0.65	300	Rebase, receipt
(d)	As (c)	0.5	300	Rejected, schedule
(e)	As (c)	0.5	at least 381	Rebase, receipt
(f)	As (c), head advances during rebase	0.65	300	Passes evaluation at its parent, superseded at the swap, then a retry against the new head
(g)	Digest matches no committed version	any	any	Rejected, defect
(h)	Non-rebasable, lag $6 > \tau_{\max}$, small updates	any	any	Rejected on the lag test, defect

Table 14.1: The conditions of Run T2, with the numerical values of the model example of Chapter 12, whose attempts S1, S2, and S3 correspond to conditions (b), (c), and (d). They are replaced by measured values before the run.

every rebase and every supersession the trace records the version of the shard the worker read, the parent the attempt named, the head found at the swap, and, for the retry, the lag and certified drift computed against the new head. *Passes when.* The measured discrepancy never exceeds $L_i \hat{\Delta}$ computed with the declared L_i, and if it does the run is reported as having falsified the declared constant. The decision and class in each condition match Table 14.1. In every rebase condition the new proposal carries the digest of the current shard and the receipt records a rebase and not a stale application. The measured total of read, recomputation, transfer, and latency is within the tolerance of the manifest of the modelled total, and the decisions follow the model in both comparisons that Table 14.1 contains. Between (c) and (d) only the deadline changes, from 0.65 s to 0.5 s, and the decision flips from receipt to schedule rejection. Between (d) and (e) only the reservable capacity changes, to at least 381 Gb/s, and the receipt returns. Any disagreement with the model is reported with its size as a model error. In condition (f) the rebased proposal carries the digest of the shard at version 40 and names the parent $(40, h_{40})$. The record shows that gluing, the invariant, and the schedule all passed against that parent, the competing commit's receipt is the head found at the swap, and the attempt is counted in n_x and in no rejection class, so that a proposal superseded after valid evaluation is distinguishable from one that failed evaluation. The retry is a new attempt against the head $(41, h_{41})$ with lag 1 and drift equal to the update norm on the competing receipt times L_i. It is admitted if the contract holds for those values and rebased again, with the costs recharged to its own deadline, if it does not. In condition (g) the proposal is rejected as a defect with the digest named. In condition (h) the rejection names the lag test, and the product $L_i \hat{\Delta}$ is reported although it is below the tolerance.

Run T3, interruption and recovery. *Initial checkpoint.* A committed checkpoint at epoch e from a T1 run, identified by its receipt digest. The uninterrupted continuation from it for E epochs is run first and serves as the reference. *Varied.* The point of interruption

as a fraction of the interval since the last committed boundary, at 0.1, 0.5, and 0.9. The scope of the failure, either one worker or a whole module. For a module failure, whether a replacement satisfying the original shape is available. *Injected.* A process kill of the worker, or loss of power to the module, at the stated point. In one further condition the sender is killed in the middle of an exchange. *Recorded.* The epoch from which the replacement resumed, the replayed fraction ρ, the time to resume, the comparison with the reference under the outcome class declared in the manifest, and κ, κ_{dec}, and θ over the window after recovery. When no satisfying replacement exists, the suspension record. *Passes when.* The replacement resumes from the last committed boundary, the receipt it names being the last receipt. No epoch is skipped or duplicated in the chain. The outcome class is met, meaning that the parameter distance to the reference and the validation metric are within the tolerances of the manifest. When no satisfying replacement exists the job is suspended at its last committed boundary, the customer is notified, and no placement outside the shape or the declared alternatives is used. In a final condition the host of a module withdraws power at a known time. The manifest declares the withdrawal as a period of unavailability, and the notice is given at a time that is varied across repeats, from longer than one epoch to shorter than the time an epoch needs to complete. The run records the notice time, the withdrawal time, the epoch in progress at each, whether a commit completed before the withdrawal, the last receipt, the replayed fraction ρ, and the time to resume. It passes when one of two things is observed and recorded with its reason. Either the job commits before the withdrawal and the receipt is dated before it, or the job is suspended and then resumes from its last committed checkpoint under the declared recovery contract. In both cases no commit is made with the withdrawn module after the withdrawal, no epoch is skipped or duplicated, and the record distinguishes a scheduled withdrawal from a fault. This condition supports a conformance claim about scheduled withdrawal. It does not measure how available a host is over a season, which remains an open measurement.

14.4 Fabric runs

Run F1, contention, capacity, and retry policy. *Initial checkpoint.* The checkpoint of Run T3. *Varied.* The reservable capacity C on the contended link, at levels fixed in the manifest that include at least one below and one above the threshold C^{*}, the smallest capacity at which the fluid test of Chapter 10 passes. The size of the competing transfer, at three levels: none, small enough that the fluid test passes, and large enough that it fails. The retry policy: immediate resubmission, resubmission after a fixed backoff, and no retry. *Injected.* The competing transfer starts on the uplink used by the exchange, either when the exchange window opens or at the middle of the window, from a traffic source whose rate is held to the reservation table. *Recorded.* The single-exchange screening decision and the demand on each link and the fluid decision for every attempt, the declared exchange interval as it stands in the job history, the delivery time of every exchange against its deadline, the counters, κ, κ_{dec}, θ, the number of retries, and the time from the first rejection to the next receipt. *Passes when.* The service rejects with class schedule exactly when the fluid test is infeasible, so that its decision agrees with the model for every attempt, and each disagreement is reported and analysed as a model error. Every accepted attempt delivers its boundary data by its deadline in every repeat, and any miss is counted as a violation of the model. Each schedule rejection names the binding link and the shortfall and marks gluing and the invariant as not evaluated. The declared interval is unchanged throughout. Raising C to C^{*} changes the decision from rejection to acceptance. The comparison between retry

policies is reported as κ, κ_{dec}, and θ for each, and it is not a pass condition.

Run F2, redundant coordinators and stale-parent races. *Initial checkpoint.* The checkpoint of Run T3. *Varied.* The number of coordinators $k \in \{1, 2, 3\}$ that submit identical valid evidence against each head, and the offset between their submissions, which is zero in one condition and a fixed small value in another. *Injected.* Nothing. The fabric is uncontended. *Recorded.* The counters, n_x in particular, the rejections by class, κ, κ_{dec}, θ, and for every superseded attempt the current head returned by the swap. *Passes when.* Exactly one receipt is issued per head. Every other attempt against that head is superseded and returns the current head. Superseded attempts appear in no rejection class. The values of n_d, n_i, and n_s equal those of the $k = 1$ baseline within the spread of repeats. When all evidence is valid each head sees k attempts, so κ is expected to fall as $1/k$ while κ_{dec} stays at 1, and both are reported. The throughput θ is reported for every k, and a fall in θ with k is recorded as a cost of redundancy and not as a fabric or repair fault.

14.5 Transcription runs

Run X1, chunk-local and assembled output. *Initial checkpoint.* A transcript job at head $(0, h_0)$, holding one or more recordings with a reference transcript whose name spellings are known, split into chunks of sixty seconds with ten seconds of overlap, together with the chunk-local recognition output stored as the checkpoint. *Varied.* The assembly policy: none, which is the chunk-local output; assembly with selection only; and assembly with selection and regeneration. The evidence standard is held fixed. *Injected.* Nothing. *Recorded.* Exact name accuracy, the number of incorrect corrections, defined as corrections to a spelling that differs from the reference, the rate of unresolved names, the number of regenerations with their GPU time and transfers, the counters, and θ. *Passes when.* The transcript invariant holds for every committed transcript. Every canonical change from a first-pass rendering names evidence meeting the standard and its source. Every unresolved mention has a candidate set containing its first-pass rendering. The accuracy, the incorrect corrections, and the unresolved rate are reported separately for each configuration, since one can rise while another falls.

Run X2, controlled mentions. *Initial checkpoint.* A constructed recording in which mentions are placed in chunk overlaps by design. The first mention has an entry in a glossary that belongs to the manifest, and that entry, Jensen Huang, is the bias supplied to any regeneration of chunk 14. The second mention is a fictional name, labelled as fictional in the manifest, followed by a relational aside of the form of Chapter 13. *Varied.* The repair policy, selection only or selection with regeneration. The evidence for the second mention, the aside alone or the aside with direct corroboration from the glossary. The first mention's n-best lists across the overlap, sharing an entry or disjoint. *Injected.* A delay of one chunk worker beyond the deadline, by a stated amount. In a second condition the same worker is released earlier. In a third the reservable capacity is raised to a high level while the delay remains. *Recorded.* The log of operations, with selection and regeneration as separate entries, each regeneration with its provenance, its supplied context or glossary, and its charged cost. The rejection classes. The class of evidence used for every canonical spelling. The unresolved sets. The receipts. *Passes when.* Disjoint lists under selection only are rejected as a defect. The same lists under a policy that authorizes regeneration produce a regeneration entry separate from selection, with provenance and charged cost, and the outcome of the regeneration is recorded without being assumed, so that a re-

decode whose list lacks the glossary entry leaves the defect unrepaired and is reported as such. A proposal that canonicalizes the second mention on the aside alone is rejected as an invariant violation, the variant that leaves it unresolved commits with its candidate set, and the variant with direct corroboration commits with a canonical spelling. The delayed worker produces a schedule rejection that persists when the capacity is raised, and the attempt is accepted when the worker is released earlier.

Run X3, interruption of transcription. *Initial checkpoint.* A head at which recognition of chunks up to some index is committed, with decoding configured to be deterministic under the supported runtime. The uninterrupted continuation is run first as the reference. *Varied.* The point of the kill within a chunk decode, early, middle, or late, and a kill during the assembly exchange. *Injected.* A process kill of the worker at the stated point. *Recorded.* The replayed fraction ρ, the output hashes, and the counters. *Passes when.* The output hash of the recovered run equals that of the reference, and the replacement resumes from the last committed boundary.

14.6 Reporting

Each run is published with its manifest, counters, receipt chain, rejection and supersession records, and the quantities named above for every repeat. A run that fails is published with its records, and the pass criteria are not revised afterward.

Each condition is repeated at least three times, and the values from every repeat are reported together with their range. Conditions are not pooled, and a quantity that varies between repeats is reported as varying and is not replaced by its mean. Where a run compares two configurations, the report states the quantity compared, the matched condition at which it was measured, and the number of repeats on which the configurations differed in the same direction. The manifest fixes tolerances, budgets, and the validation target before the first run, so that an outcome cannot be reclassified after it is seen.

14.7 What the protocols cannot establish

The protocols test the specification and do not establish that a deployment is economical. The prototype has two modules, so effects that appear only with many modules, such as contention among several jobs for shared links, are exercised by the contended condition of Run F1 and not by scale. The constants of the model examples, namely the Lipschitz constants, reservable capacities, and shard sizes, are replaced by measured values before each run and are valid only for the measured configuration. The runs hold the host of each module constant and do not vary the heat demand, the power terms, or the season of a host site, except through the scheduled withdrawal of Run T3, which tests what the service promises at a known withdrawal time and not how often a host withdraws, so they say nothing about whether small modules at heat-consuming sites are worth operating. Finally, the transcription runs use a controlled recording with a fictional name, and their accuracy figures describe that construction and do not predict the accuracy of a deployed transcription service.

Chapter 15

The Procurement Decision

The first procurement decision is taken from the measurements of Chapter 14. Two bounds convert the measurements into a statement about what faster optics could buy. Let φ be the exposed communication fraction, the fraction of elapsed time during which the job's progress is held up by boundary data that have not yet arrived, so that every module that has finished its local work is waiting. It is measured from the traces of Runs T1 and F1.

Proposition 15.1. *Hold the workload, the placement, the synchronization schedule, and the remaining execution fixed, and suppose the exposed communication time were removed entirely. Then the committed epochs per unit time rise by a factor of at most* $1/(1-\varphi)$.

Proof. The elapsed time T contains φT of exposed communication. With everything else fixed, removing it leaves $(1-\varphi)T$ for the same committed epochs, since the remaining execution is unchanged by hypothesis, so the rate rises by exactly $1/(1-\varphi)$ in the limit and by less when some of the time cannot be removed. □

Proposition 15.2. *If every schedule rejection of a run could be turned into a receipt and the other outcomes were unchanged, the decided yield would rise from* κ_{dec} *to at most* $(n_r + n_s)/(n_a - n_x)$.

Proof. The number of decided attempts $n_a - n_x$ is unchanged, and the receipts increase by at most n_s. □

Both bounds are ceilings for a particular thought experiment, in which the fabric improves and the measured workload and execution do not change. They do not predict what a redesigned run would do. Faster optics could make placements admissible that the present fabric refuses, and could change how often a failure is replayed or which shapes are worth requesting. Those effects are real and are not in the bounds, so they belong in a measured comparison, in which the protocols of Chapter 14 are run again on the faster fabric with the placement and synchronization chosen for it. The bounds are useful for the decision because they are cheap and conservative in one direction: each is useful when it is small. If φ is small, eliminating exposed communication offers little throughput gain under the fixed execution measured here, and if n_s is a small part of $n_a - n_x$, so are yield gains from removing schedule rejections alone. The second bound need not be attained, because a schedule rejection leaves gluing and the invariant unevaluated, and the count alone does not show whether every such attempt would pass them. If the rejections are mostly defects or invariant violations, by Proposition 8.2 no change to the fabric can remove them, and the remedy lies in authorization, contract, or placement. When both bounds are large enough to matter, the next step is to price a faster fabric and compare pluggable and co-packaged options on energy per committed epoch, on repair time, measured for pluggables

by replacing a transceiver during Run F1, and on throughput under load, measured by the contended conditions of Run F1. Co-packaged optics enters the comparison through vendor data and any hardware available for loan, and availability and serviceability must be verified for the specific deployment, since industry reporting identifies optical engine yield, silicon photonics capacity, and advanced packaging as supply bottlenecks [5]. The design choice made here is to put photonics at the scale-out boundary and use shipping GPU platforms inside each module, so that the cloud architecture can be tested before any commitment to custom packaging. Availability and lead time of pluggable optics and of accelerators also depend on a supply landscape that analyst reports track [15] and that this monograph does not model, so the decision should record the source of each supply assumption it uses. These bounds concern optics only. Whether small modules at heat-consuming sites are worth operating is a separate question that the measurements here do not settle, because it depends on the heat demand and the power terms of the hosts, which the protocols do not vary.

Appendix A

Conformance Clauses

A service conforms to this specification when it satisfies the requirements of Chapters 4 through 13. Conformance is stated per job class. A service may claim conformance for shape-first allocation and constrained replacement without claiming it for result records, in which case the claim names the chapters satisfied. The procedures below give one test for each requirement. Each test names the stimulus, the observation, and the condition for passing, and may be run against the prototype of Chapter 14 or against any other implementation.

Test 4.1, shape-first allocation. Submit one request with an explicit shape and one stating only a device count. Pass when the second is recorded with the default shape of Requirement 4.1 and the first is recorded as stated.

Test 4.2, reported placement. Place a job on part of a rack and request the placement report. Pass when no reported committed figure describes a scope larger than the one allocated, and every figure is labelled with its scope.

Test 4.3, constrained replacement. Fail a module while no module satisfying the original shape is free. Pass when the job is suspended at its last committed boundary, the customer is notified, and no replacement placement outside the shape or the declared alternatives is used. Repeat with a satisfying module free and pass when it is chosen. Repeat once more with the host of a module giving notice of a power withdrawal at a known time, and pass when the job either commits before the withdrawal or is suspended and resumes from its last committed checkpoint, with the notice time and the withdrawal time recorded.

Test 5.1, declared exchange. Attempt a cross-module transfer not covered by any contract. Pass when the transfer is refused or logged as unsanctioned.

Test 5.2, validation on arrival. Inject an exchange whose payload violates the validator. Pass when the failure is recorded with its step and reported to the job, and the number of retries does not exceed the contract limit.

Test 5.3, interval as a parameter. Throttle the fabric during a run. Pass when the exchange interval in the job history is unchanged and any lateness appears as a recorded delay rather than as an altered interval.

Test 5.4, feasibility at admission. Submit a job whose declared interval, exchanged size, and committed bandwidth violate the inequality of Requirement 5.4. Pass when the job is refused and the refusal names the boundary and the failing quantity. Repeat with an overlapping declaration and pass when the report states that the test is weaker.

Test 6.1, identified boundaries. Request a checkpoint at a step not declared as a synchronization boundary. Pass when it is deferred to the next boundary or refused.

Test 6.2, resume from boundary. Kill a worker between boundaries. Pass when the replacement starts from the last committed boundary, the uncommitted work is regenerated, and the replayed fraction ρ is recorded.

Test 6.3, outcome class. For a deterministic workload run once uninterrupted and once with an interruption, compare output hashes. Pass when the delivered class matches the declared class: equal hashes for the deterministic class, or comparison within the declared tolerances for the training class. The report shall name the class actually satisfied.

Test 7.1, preserved candidates. Run an assembly stage that changes a result. Pass when the record holds the original candidate and its input span for every change.

Test 7.2, evidence for correction. Inspect the record for each change. Pass when each carries the evidence segment and the rule or component that proposed it.

Test 7.3, unresolved status. Supply an input in which the available evidence supports two candidates equally. Pass when the item is marked unresolved with both candidates retained and no unrecorded rule selects between them.

Test 8.1, declared interface content. Submit a job with an interface that carries only a digest and no declaration about summaries. Pass when the submission is refused or the interface is flagged as a summary interface in the job record.

Test 8.2, explicit global invariants. Submit a job with a global power budget that is not declared as an invariant. Pass when the service requires the declaration, and when declared, evidence for the invariant appears in every commit.

Test 8.3, obstruction reporting. Construct a pair of proposals whose defect lies outside the image of the repair operator, and a second pair whose defect lies inside it. Pass when the first is reported as unrepairable and is not resubmitted unchanged, and the second is reported with its repair.

Test 9.1, atomic head update. Submit two valid commits against the same parent from separate coordinators. Pass when exactly one receives a receipt and the other receives the stale outcome with the current head, recorded as a supersession under Requirement 11.1.

Test 9.2, verifier statement. Request the published verifier statement for each declared invariant class and inspect a receipt. Pass when the receipt names a verification level that appears in the statement.

Test 9.3, receipt content. Walk the chain of receipts of a completed job. Pass when each names the job, epoch, parent digest, and manifest digest, references its checkpoint, and the parent digests link without gaps.

Test 10.1, reservable capacity. Compare the capacity reported for a link with the line rate of the optics on it. Pass when the reported figure is the reservable one and its basis is stated.

Test 10.2, placement by feasibility. Submit two exchanges that share a link and whose combined demand exceeds the link capacity within their deadlines, each of which would pass the single-exchange test alone. Pass when the placement is refused. Repeat for a replacement placement after a module failure.

Test 11.1, rejection record. Cause one rejection of each class against the same head, with the schedule rejection caused by a predicted infeasible exchange. Pass when each produces a record naming the parent head, the class, its evidence, and a remedy, the schedule record marks gluing and the invariant as not evaluated, the head is unchanged afterward, and the job tallies of attempts, retries, rejections by class, supersessions, and receipts match the events and reconcile.

Test 11.2, supersession. Submit the same valid evidence from two coordinators against one head. Pass when exactly one receives a receipt, the other receives a supersession record naming the current head and the winning attempt, the superseded attempt is counted in no rejection class, and the reconciliation $n_a = n_r + n_d + n_i + n_s + n_x$ holds.

Test 12.1, base digest. Submit a proposal carrying a digest that matches no committed version. Pass when it is rejected as a defect with the digest named.

Test 12.2, certified drift. Commit several epochs with known update norms and submit a proposal read at an earlier version. Pass when the drift computed by the service equals the sum of the recorded norms between the two versions, with no earlier shard version retained.

Test 12.3, two-part admission. Submit one proposal with a lag within the limit and drift above the tolerance, and another with a lag above the limit and drift below the tolerance. Pass when both are rejected, each rejection names which test failed, and the product of the Lipschitz constant and the certified drift is reported.

Test 12.4, admissibility of rebase. Submit a stale proposal from a module whose authorized repairs include rebase and another from a module whose authorized repairs do not. Pass when the first is rebased and committed with the repair recorded in the receipt, and the second is rejected as a defect.

Test 13.1, declared evidence standard. Propose a canonical spelling supported only by a relational aside. Pass when the attempt is rejected as an invariant violation and the aside appears in the record as a narrowing clue.

Test 13.2, declared repair policy. Submit overlapping chunk results with disjoint n-best lists under a selection-only policy and then under a policy authorizing regeneration. Pass when the first is rejected as a defect, and in the second any regeneration appears as an operation separate from selection, with provenance and a cost charged to the deadline.

Test 13.3, unresolved status is committable. Submit a transcript in which one mention has insufficient evidence and is marked unresolved with its candidates. Pass when the attempt commits and the receipt reports the counts of canonical corrections and unresolved mentions.

A.1 Stating a conformance claim

A conformance claim names the version of this specification, the job classes it covers, and the chapters it satisfies. It states the verification levels of Requirement 9.2 that the verifier supports and the outcome classes of Requirement 6.3 that the service can report. It lists each test that was run, with the manifest digest of the run, the date, and the result, and it lists each test that was not run or that failed. A claim about scheduled withdrawal rests on the withdrawal repetition of Test 4.3 and the withdrawal condition of Run T3. It says what the service does when a host withdraws a module at a known time, and it says nothing about how often hosts withdraw modules over a season. A claim of partial conformance is a claim about the tests listed and carries no implication about the others. A service that passes the tests for shape-first allocation and constrained replacement, for example, has said nothing about its result records.

A claim is withdrawn or restated when the service changes in a way that could affect a listed test, such as a new verifier, a new placement policy, or a new fabric. Tests whose results depend on measured constants, such as Test 10.1 and Test 12.3, state the measured values used, so that a later reader can tell whether a pass was obtained at a capacity or a Lipschitz constant that still holds.

Appendix B

Requirement Traceability

Each requirement is traced to the formal object that defines what it asks, the record or outcome by which a customer can observe it, the conformance test of Appendix A, and the run of Chapter 14 that exercises it. A run exercises a requirement when the run's recorded quantities and pass criteria depend on it. Requirements that apply in every run through the common conditions of Chapter 14, namely reconciliation of the counters, verification of the receipt chain, and the form of rejection and supersession records, are marked as all runs.

Req.	Formal object	Record or outcome	Test	Run
4.1	Communication shape	Placement record	4.1	C1
4.2	Module, scope	Placement report	4.2	C1
4.3	Constrained replacement proposition	Suspension record	4.3	T3
5.1	Exchange contract	Refusal or unsanctioned log	5.1	C1
5.2	Validity predicate	Validation record per exchange	5.2	T1
5.3	Interval of the contract	Interval in the job history	5.3	T1, F1
5.4	Single-exchange inequality	Admission refusal naming the boundary	5.4	F1
6.1	Committed checkpoint	Receipt chain	6.1	T3
6.2	Replayed fraction	Resume record	6.2	T3, X3
6.3	Outcome class	Class satisfied by the result	6.3	T3, X3
7.1	Mention record, preserved candidates	Original candidate and span	7.1	X1, X2
7.2	Evidence standard	Evidence on each correction	7.2	X1, X2
7.3	Unresolved status	Candidate set	7.3	X1, X2
8.1	Interface cells of the partition complex	Job declaration	8.1	C1
8.2	Dependent sum with the invariant	Declaration and commit evidence	8.2	C1
8.3	Image of the repair operator	Rejection record, class defect	8.3	T2, X2
9.1	One-winner proposition	Supersession record	9.1, 11.2	F2
9.2	Ready type, verifier	Verification level on the receipt	9.2	all runs
9.3	Receipt	Receipt chain	9.3	all runs
10.1	Reservable capacity	Reservation table	10.1	F1
10.2	Fluid feasibility	Rejection record, class schedule	10.2	F1, T2, X2

Req.	Formal object	Record or outcome	Test	Run
11.1	Outcomes of a commit attempt	Rejection and supersession records, tallies	11.1, 11.2	all runs, F2
12.1	Proposal triple	Rejection record, class defect	12.1	T2
12.2	Certified drift	Update norm on the receipt	12.2	T1, T2
12.3	Staleness contract	Rejection record with $L_i\hat{\Delta}$	12.3	T2
12.4	Rebase and stale application	Operation field of the receipt	12.4	T2
13.1	Evidence standard	Evidence class of each correction	13.1	X2
13.2	Repair policy	Operation log	13.2	X2
13.3	Transcript invariant	Receipt counts of corrections and unresolved mentions	13.3	X1, X2

Two readings of the table are worth stating. Every requirement has at least one run, and the requirements that concern admission and declaration only (4.1, 4.2, 5.1, 8.1, 8.2) are exercised by the contract run alone, because no job needs to start for them to be tested. The requirements that concern commit and supersession (9.1, 11.1) are exercised in the redundant-coordinator run so that redundancy is visible in the tallies and is not attributed to the fabric or to the repair mechanism.

Bibliography

[1] Flyxion. Local Transitions and Global Admissibility. Public repository, standardgalactic, 2026.

[2] NVIDIA Corporation. NVIDIA GB300 NVL72. Product specification page, `nvidia.com/en-us/data-center/gb300-nvl72`, accessed November 2026.

[3] NVIDIA Corporation. System Hardware and Components, NVL72 AI Factory Enterprise Reference Architecture. `docs.nvidia.com/enterprise-reference-architectures/nvl72-ai-factory`, accessed November 2026.

[4] NVIDIA Corporation. NVIDIA ConnectX-8 SuperNIC Datasheet. Product datasheet, 2025.

[5] TrendForce. NVIDIA and Broadcom Begin Volume Ramp of CPO Switches, with Optical Engine Yield and Advanced Packaging Capacity Emerging as Key Expansion Bottlenecks. Press release, 27 July 2026.

[6] International Energy Agency. Energy and AI. Special report, Paris, April 2025.

[7] Data Center Dynamics. IEA: Data Center Energy Consumption Set to Double by 2030 to 945 TWh. News report, 2025.

[8] Lutken, T. If You Can't Take the Heat, Distribute It. *Vibrant Environment*, Environmental Law Institute, 11 December 2025.

[9] Ballard, M. Financial and Technical Obstacles Hit Danish Datacentre Heat Recycling Plan. *Computer Weekly*, 13 May 2021.

[10] Qarnot. District Heating and Data Centers: The Future of Energy Transition. Company blog, 22 February 2024.

[11] Data Center Dynamics. Qarnot Upgrades Its Digital Boiler with ITRenew's Refurbished OCP Hardware. News report, `datacenterdynamics.com`, accessed November 2026.

[12] Huawei. Serving Large Language Models on Huawei CloudMatrix384. arXiv:2506.12708, 2025.

[13] Tao, Z., Zhou, Y., Yu, W., Chang, H., et al. On-Chip Large-Scale All-Optical Interconnect for Ultra-Low-Latency Deep Neural Network Inference. *National Science Review*, 13(19), nwag282, 2026. DOI 10.1093/nsr/nwag282.

[14] Han, C., Yang, Q., Qin, J., et al. Exploring 400 Gbps/λ and Beyond with AI-Accelerated Silicon Photonic Slow-Light Technology. *Nature Communications*, 16, 6547, 2025. DOI 10.1038/s41467-025-61933-5.

[15] LightCounting. Vendors and Markets for Optics in China. Market report, January 2025.