Tessera: Failure-Atomic Reconfiguration of Disaggregated Memory

Anonymous Author(s)

Abstract

Memory disaggregation lets a cluster replace stranded per-server capacity with a shared pool, but it also turns memory-management operations into distributed reconfigurations. A compute node may fail after publishing a mapping but before retiring its predecessor, while a memory appliance may reboot with data intact but ownership metadata only partially updated. Existing far-memory systems either rebuild the entire pool after such failures or place a replicated metadata transaction on every allocation and migration. The first choice makes recovery proportional to pool size; the second taxes the common path.

We present Tessera, a disaggregated-memory control plane whose reconfigurations are failure-atomic without requiring consensus on memory accesses. Tessera separates a short-lived *ownership token* from persistent extent metadata, uses monotonically increasing generations to fence stale compute nodes, and records only the boundary between prepared and visible mappings. This design keeps loads and stores on the hardware data path and makes recovery proportional to the number of interrupted reconfigurations rather than installed memory. Our prototype comprises a Linux kernel module, a user-space pool manager, and firmware support for an RDMA/CXL memory appliance. On a 32-node testbed, Tessera adds 2.1% geometric-mean overhead to applications with working sets up to 1.5 TB. It restores service after compute, manager, and appliance failures in 184–691 ms, 18–64× faster than scanning the pool, while retaining 91% of ideal packing efficiency. These results show that strong ownership semantics can be moved off the memory-access path without surrendering bounded recovery.

CCS Concepts: • **Software and its engineering** → **Distributed systems organizing principles**; **Operating systems**; • **Information systems** → *Storage management*.

Keywords: disaggregated memory, failure recovery, reconfiguration, ownership, RDMA, CXL

ACM Reference Format:
Anonymous Author(s). 2026. Tessera: Failure-Atomic Reconfiguration of Disaggregated Memory. In *ACM Symposium on Operating Systems Principles (SOSP)*. ACM, New York, NY, USA, 7 pages.

1 Introduction

Fast fabrics have made memory disaggregation a plausible alternative to provisioning DRAM for each server's peak. Systems such as Infiniswap, LegoOS, and AIFM show that remote memory can support unmodified or lightly modified applications at useful performance [8, 14, 15]. Emerging CXL fabrics go further by permitting processors to load from pooled memory through coherent, hardware-managed paths [5]. In both settings, the attractive data path is simple: once a mapping is installed, a compute node accesses memory without consulting a centralized service.

The control path is not simple. Allocating, migrating, or reclaiming pooled memory changes three pieces of state: the allocator's ownership record, the memory appliance's access-control entry, and the compute node's page table. They reside in different failure domains and become visible in different orders. A crash in the middle may leave two nodes able to name the same physical extent, leave an extent unreachable, or resurrect a mapping whose contents have already been reassigned. These are not merely leaks. A stale writer can silently corrupt another tenant's memory after the fabric has resumed service.

The usual remedies are poorly matched to memory. One remedy reconstructs ownership by scanning the pool and all surviving page tables. Recovery then grows with installed capacity, even when only one 2 MB extent was moving. Another remedy commits every mapping transition through a replicated log. Although appropriate for control-plane decisions, a round of consensus for short-lived mappings amplifies allocation latency and manager load. Replicating the data itself, as in in-memory storage systems [6, 12], changes both the cost and the semantics of a capacity pool.

Tessera starts from a narrower question: *what is the minimum durable information needed to make a mapping change failure-atomic?* Its answer is a small prepared record containing the old owner, new owner, and next generation. The record does not describe every page. It marks the one transition whose visibility may be ambiguous. A memory appliance grants access only to a signed token containing that generation. Before reusing an extent, the manager advances the appliance's generation; all older tokens become unusable even if a partitioned compute node later returns. Once the new mapping is published, the manager converts the prepared record to a compact stable owner entry.

This separation creates a useful asymmetry. Allocation, migration, balancing, and failure handling use replicated control state, but ordinary loads, stores, and remote reads

remain entirely outside consensus. At recovery, the manager examines the bounded set of prepared records and asks appliances which generation is authoritative. It does not scan application data or enumerate all page tables.

This paper makes four contributions:

- We identify the *reconfiguration gap* in disaggregated memory: the interval in which durable ownership and hardware reachability disagree.
- We design a token and generation protocol that closes this gap while preserving a one-sided, consensus-free memory-access path.
- We give recovery rules and invariants covering compute-node, manager, and memory-appliance failures, including delayed nodes and repeated failures during recovery.
- We implement and evaluate Tessera on RDMA hardware with a CXL-like access-control shim, showing sub-second recovery and low steady-state cost.

2 Background and Motivation

2.1 System setting

We consider a rack-scale pool composed of compute nodes and memory appliances. An appliance exports fixed-size 2 MB *extents*; a compute node may map 4 KB pages within an extent. The data path is either CXL-style load/store or one-sided RDMA. As in prior far-memory work [1, 2], the CPU on the memory appliance is not involved in ordinary accesses. A replicated pool manager assigns extents, issues access tokens, and balances capacity. It is not on the load/store path.

Our fault model allows crash-stop compute nodes, manager replicas, and memory appliances; messages may be delayed, reordered, or duplicated. An appliance's DRAM is lost on power failure, but its small nonvolatile generation table survives restart. The network may partition and later heal. We assume a majority of manager replicas remains available and use crash-tolerant consensus for manager metadata [11]. Byzantine behavior and malicious firmware are outside our model. Applications are responsible for durability of their data; Tessera protects ownership and isolation, not application contents.

2.2 The reconfiguration gap

Consider moving extent e from compute node A to B. Four actions are necessary: stop A, copy live pages, authorize B, and publish B's mapping. No order makes crashes harmless. Revoking A first can make the only live copy unavailable. Publishing B first permits concurrent writers. Updating only the manager first leaves hardware enforcing old authority; updating hardware first leaves durable metadata describing the old state.

This interval is the *reconfiguration gap*. It appears in more operations than migration: allocating an extent from a free list, reclaiming memory from a failed node, changing tenant keys, and evacuating an appliance all cross it. Traditional distributed file systems close related gaps with locks and logs [16], while persistent-memory systems order data and metadata writes within one machine [17, 18]. A disaggregated-memory mapping spans independent machines and a hardware access control boundary, so neither technique alone is sufficient.

2.3 Why scanning and synchronous logging fall short

Our measurements of a straightforward rebuild illustrate the scale mismatch. Reading a 16-byte owner record for every 2 MB extent in a 16 TB pool takes 11.8 s with batched RDMA; validating mappings at 32 compute nodes raises this to 16.4 s. The time is paid even if the failed manager interrupted no operation. Conversely, synchronously appending a replicated record for every 4 KB mapping raises the median fault-service time from 7.3 to 31.6 μs and saturates the leader near 2.1 million mappings/s. The result is a false choice between slow recovery and an expensive common path.

Design goals. Tessera therefore targets four properties: (1) exclusive ownership despite arbitrary crash points; (2) recovery proportional to incomplete operations; (3) no manager or consensus traffic on ordinary memory accesses; and (4) incremental deployment on appliances with a small durable table. It does not make volatile remote memory persistent, mask application bugs, or provide transactions across application objects.

3 Design

3.1 Overview

Figure 1 shows the architecture. Each compute node runs a kernel client that requests extents, installs mappings, and caches signed tokens. Three manager replicas maintain a replicated transition log and a compacted ownership map. Each memory appliance contains a *gate*: a small access-control unit that checks the extent identifier, tenant, and generation carried by a token. The gate's generation table is nonvolatile; application data remains ordinary volatile DRAM.

The central object is an *ownership token*

$$\tau = \mathrm{Sign}_{K_a}\left(\langle e, g, o, t_{\mathrm{exp}}, r\rangle\right), \tag{1}$$

where K_a is an appliance-specific key, t_{exp} is an expiration time, and r is a permission mask. A gate accepts τ only if its signature is valid, it has not expired, and g equals the durable generation $G_a[e]$. Tokens are capabilities, but generations—not wall-clock leases—provide the safety boundary. Expiration bounds resource retention and key exposure; it is not needed to prevent a stale write after reuse.

3.2 State and invariants

For each extent, manager metadata is in one of four states:

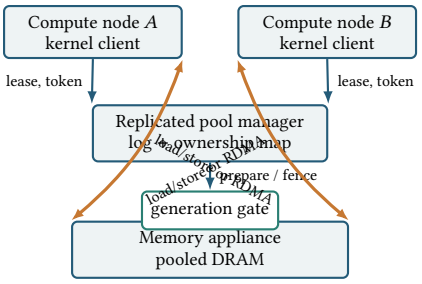

Figure 1. TESSERA keeps the replicated manager off the data path. The memory appliance gate accepts an access only when its token generation matches the durable generation for the target extent.

State	Meaning
FREE	no compute node may access the extent
OWNED(o, g)	owner o may use generation g
PREPARED$(o, n, g{+}1)$	transition from o to n is incomplete
LOST(g)	contents unavailable; generation retained

The appliance stores only $G_a[e]$ and a tenant-key identifier. Stable owner state is compacted in the manager; only PREPARED records must remain in the replicated log. The protocol maintains two invariants.

Invariant 1 (Exclusive authority)**.** *For every extent e, all accesses accepted by its gate at a logical instant name the same owner and the gate's current generation $G_a[e]$.*

Invariant 2 (Recoverable boundary)**.** *If manager metadata and the appliance generation disagree for e, the manager contains a durable PREPARED record that identifies both adjacent generations and both candidate owners.*

The first invariant prevents stale access; the second makes recovery local. Together they avoid a global scan. Notice that duplicated or delayed messages cannot roll a generation backward: the appliance implements advance(e, g') only when $g' > G_a[e]$.

3.3 Failure-atomic transition

Changing ownership from A to B follows four phases, illustrated in Figure 2.

1. **Prepare.** The manager commits PREPARED$(A, B, g{+}1)$ to its replicated log. It stops renewing A's lease and asks A to quiesce writes.
2. **Transfer.** If data must survive the move, A or a copy engine transfers live pages to a staging extent. Dirty-page tracking permits one final bounded copy after quiescence.
3. **Fence and publish.** The manager advances the appliance to $g{+}1$. This single durable write invalidates every token issued for g. It then issues B a token and waits for B to install the mapping.

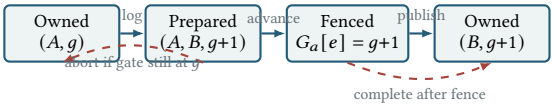

Figure 2. Ownership transition and recovery choices. Before the generation advances, recovery may abort; afterward it must complete the transition. No recovery edge moves the appliance generation backward.

4. **Stabilize.** After B acknowledges publication, the manager replaces PREPARED with OWNED$(B, g{+}1)$ and releases any staging extent.

Quiescence improves availability but is not trusted for safety. A failed or partitioned A may continue issuing requests. Once the gate durably advances, those requests carry g and fail. Likewise, issuing B's token only after the advance means A and B never possess simultaneously valid generations.

Lemma 1 (No stale reuse)**.** *After an extent transitions from generation g to $g{+}1$, no access using a token issued before the transition is accepted.*

Proof. Every pre-transition token names a generation at most g. The gate accepts only a token whose generation equals its durable entry. The transition changes that entry monotonically to $g{+}1$ before issuing a new token; retries and restarts cannot decrease it. Therefore no pre-transition token can satisfy the gate check. □

3.4 Recovery

A newly elected manager first restores its compact ownership snapshot and then replays the suffix of the transition log. Stable OWNED extents need no work. For each PREPARED record it reads the appliance generation:

- If $G_a[e] = g$, fencing never became durable. Recovery may abort the operation, renew A, and delete staging state. If A failed, it instead continues the transition to a replacement owner.
- If $G_a[e] = g + 1$, old tokens are already invalid. Recovery reissues an idempotent token to B, asks B to install or confirm the mapping, and stabilizes the record.
- If the appliance is unavailable, the extent remains PREPARED and is excluded from allocation. Other extents recover independently.
- If the appliance reports a larger generation, a later transition was committed by a previous leader. Recovery replays forward from the log and never restores the older owner.

Repeated failures are harmless because every action is idempotent: log entries have unique operation identifiers, gate advances are monotonic, token issuance is deterministic for an operation, and mapping acknowledgments include the

generation. With q prepared records and p available appliances, recovery requires $O(q)$ metadata reads in $O(\lceil q/p \rceil)$ parallel rounds. Pool capacity does not appear in the bound.

Compute-node failure. Failure detection stops lease renewal and prevents new allocations. The manager advances each extent formerly owned by the node before placing it on a recovery queue. Applications that replicate data can attach a replacement and continue; otherwise the extent becomes Lost. In both cases, isolation is restored before reuse.

Memory-appliance failure. When volatile contents disappear, the persistent generation table prevents old tokens from becoming valid after reboot. The manager increments generations in batches and marks extents Lost; application replicas or checkpoints reconstruct data. An appliance that merely restarts without losing DRAM can rejoin after the manager validates its boot identifier and generation digest.

3.5 Leases, time, and partitions

Leases allow the manager to reclaim metadata for unreachable nodes without retaining unbounded session state. They are deliberately not the fencing mechanism. Clock skew can make two parties disagree about whether a lease has expired, but it cannot make the gate accept a mismatched generation. During a manager partition, a minority cannot commit Prepared records or issue new generations. Existing mappings keep working until their token expiry; clients renew through a majority. We set token lifetime to 30 s and renew at 10 s in the prototype, trading partition tolerance against revocation latency for extents that are never explicitly fenced.

3.6 Compaction and metadata cost

Stable ownership is encoded as sorted runs of adjacent extents with the same owner and generation epoch. A background compactor snapshots these runs and truncates log prefixes after all replicas install the snapshot. Prepared records are pinned until stabilization. The in-memory index is 24 bytes per extent; the appliance needs an 8-byte generation and 4-byte key identifier. With 2 MB extents, this is 0.0017% raw metadata before replication. Fine-grain 4 KB mappings remain local to the compute node and need not enter manager state.

4 Implementation

Our prototype contains 18.7 K lines of Rust and 3.9 K lines of C. The manager is a Rust service whose three replicas use a Raft log. A sharded extent map lets unrelated transitions proceed concurrently; each shard has a 64-bit epoch to detect a stale leader. We batch up to 128 prepares or 256 stabilization records per log append. Snapshots use copy-on-write runs, so compaction does not pause allocations.

The Linux 6.8 client is a kernel module exposing a `/dev/tessera` device. It maps registered RDMA regions into a process or backs a private `mmap` region with demand paging. A per-mm radix tree stores local 4 KB-to-extent offsets. Migration write-protects the source PTEs, drains CPU stores with a TLB shootdown, copies dirtied pages, and acknowledges quiescence. The module caches tokens per extent and renews them asynchronously; a data-path operation never waits for renewal while a current token remains valid.

Our appliance is an x86 server with 512 GB of DRAM and a Mellanox ConnectX-6 NIC. An eBPF/XDP shim validates a compact token identifier on the first packet of an RDMA region and programs an NIC memory key scoped to the extent. This emulates the generation check expected from a CXL fabric manager. The durable generation table resides on an Corex Optane P1600X device; group commit flushes up to 64 adjacent updates. A gate advance takes 8.4 μs median and 19.7 μs at p99. CXL-native hardware should remove the control packet but is not required for the protocol.

Engineering around wraparound. Generations are unsigned 64-bit integers and never reset when an extent becomes free. At one million transitions per second on one extent, wraparound would take more than 584,000 years. We nonetheless treat reaching $2^{64} - 1$ as an appliance retirement condition rather than defining wrap semantics.

5 Evaluation

We ask four questions: (1) What steady-state overhead does Tessera impose? (2) How quickly does it recover from failures? (3) Does its metadata and recovery work scale with incomplete transitions rather than pool capacity? (4) What utilization does safe reconfiguration permit?

5.1 Methodology

Our testbed has 32 dual-socket compute servers, each with two 24-core AMD EPYC 7402 CPUs and 128 GB local DRAM, plus eight 512 GB memory appliances. Two 100 Gb/s RoCEv2 links connect each machine to redundant switches. We cap local memory so 25–75% of each workload's working set is remote. Manager replicas run on three separate servers. Reported results are medians of seven 10-minute runs; error bars show the 5th and 95th percentiles.

We use five workloads: Redis 7.2 with 64 million 1 KB values; RocksDB 8.9 with a 480 GB database and YCSB-A; Graph500 breadth-first search at scale 29; a PyTorch recommendation model with a 1.1 TB embedding table; and a synthetic allocator that repeatedly maps, touches, and frees extents. The applications use the same local mapping API under all systems.

We compare against RecoverAll, which uses identical data-path code but rebuilds ownership by scanning appliances and client maps after a control-plane failure, and SyncLog, which commits every 4 KB mapping change to the

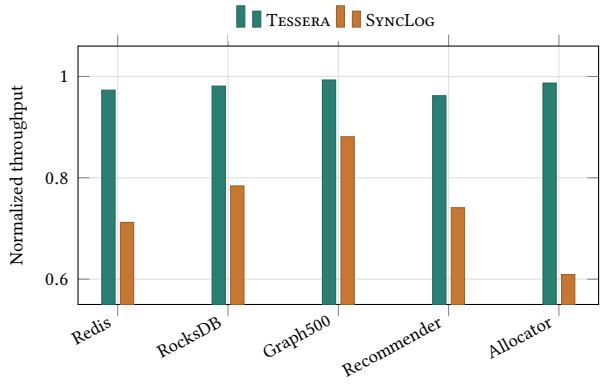

Figure 3. Application throughput normalized to the same remote-memory data path without failure-atomic control. Tessera keeps transitions at extent granularity and stays within 3.8% of the baseline.

Table 1. Recovery time under injected failures. Values are median / p99. RecoverAll scans 4 TB of installed memory; Tessera resolves 1,024 prepared transitions unless noted.

Failure	RecoverAll	Tessera
Manager leader	11.8 / 13.6 s	184 / 267 ms
Compute node	14.2 / 18.1 s	391 / 614 ms
Appliance soft reboot	12.0 / 14.4 s	438 / 691 ms
Failure during recovery	16.7 / 21.3 s	622 / 948 ms

replicated log. Local fits the working set in local DRAM and is an upper bound, not a capacity-equivalent configuration.

5.2 Steady-state overhead

Figure 3 reports throughput normalized to a version with the same remote-memory data path but no failure-atomic control plane. Tessera sustains 96.2–99.3% of that throughput (2.1% geometric-mean overhead). Redis and the recommendation model are most sensitive because they allocate during the run; Graph500 performs almost all control operations during initialization. In contrast, SyncLog loses 12–39% by logging fine-grain mappings and shows a 4.3× increase in p99 allocation latency. Token checking adds 83 ns to the first access that binds an RDMA memory key and is absent from subsequent accesses.

5.3 Recovery latency

We inject failures after a uniformly random transition phase while running RocksDB and the allocator. Recovery ends when new allocations and affected mappings are available. Table 1 summarizes the results.

A manager failure is dominated by leader election and two parallel batches of generation queries. Compute-node recovery additionally fences 2,048 owned extents. The appliance reboot includes rebuilding NIC keys from the durable

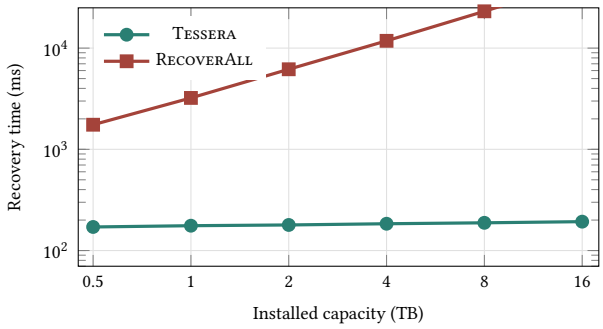

Figure 4. Manager recovery with 1,024 prepared records. Tessera tracks the ambiguous transitions, so its work is independent of installed capacity; scanning grows linearly.

generation table. A second failure during recovery repeats idempotent work but does not trigger a pool scan. Across all cases, we observe no accepted request with a stale token in 4.1 million injected transition failures.

Recovery causes a short availability dip but little cluster-wide disruption. For a failed compute node, unaffected Redis clients lose 3.6% throughput for one 100 ms interval; p99 latency returns to its pre-failure range within 620 ms. RecoverAll blocks new allocations for the entire scan and loses 19–27% throughput during its recovery window.

5.4 Scaling with ambiguity, not capacity

Figure 4 varies installed pool size while holding 1,024 incomplete transitions, then varies transitions in a fixed 4 TB pool. Tessera recovery is nearly flat as capacity grows from 512 GB to 16 TB (171–193 ms). It grows linearly with prepared records until parallel gate-query batches saturate one manager core: 84 ms for 64 records, 612 ms for 16,384. RecoverAll instead grows linearly with installed capacity and is insensitive to ambiguity.

The ownership snapshot is 31.4 MB at 4 TB and 122.8 MB at 16 TB. Restoring it is overlapped with leader election; its sequential read takes 14–51 ms. At 64 compute nodes, the manager sustains 3.8 million extent operations/s with batching and 0.92 million without. Gate storage is 3.0 MB per 512 GB appliance.

5.5 Packing efficiency and migration

Safe fencing would be of limited use if coarse extents stranded capacity. Under a 24-hour trace replay derived from the five workloads, best-fit placement without failures reaches 94% pool utilization. Tessera reaches 91%; the gap comes from staging extents and prepared records withheld during transitions. Limiting each tenant to 128 concurrent transitions caps unavailable capacity at 256 MB and changes migration completion time by less than 7%.

Live migration of a 2 MB extent takes 241 μs median when 10% of pages are dirty and 1.7 ms when every page is

dirty. Generation advancement contributes 8.4 μs. Disabling copy pipelining raises the all-dirty case to 2.9 ms; disabling batched stabilization adds 17% manager CPU. These ablations support the design choice to keep safety at extent granularity while tracking liveness and copying at page granularity.

6 Safety Argument

We model one extent because transitions on different extents commute. Manager log commitment provides a total order of PREPARED records. The gate's advance operation provides a monotonic order of generations. The only step that creates authority for a new owner is token issuance, which occurs after both orders contain the transition. Thus, if owner B has a valid token for $g + 1$, the gate has already rejected every token for g or below.

Crash points partition into two cases. Before the gate advance, the durable PREPARED record lets recovery return to A or retry; no B token is valid. After the advance, A is fenced and recovery can only move forward to B. A crash between advance and token issuance temporarily leaves no owner but never two owners. This is the same safety-over-availability choice made by fencing-token protocols [4], applied at the memory fabric.

The argument assumes a gate does not lose or roll back its durable generation. We protect the table with checksummed, double-buffered 4 KB pages and a monotonic boot counter. A torn page selects the older valid copy and replays its journal; if neither copy validates, the appliance refuses all access and requires manager-assisted reinitialization. This fail-closed behavior can reduce availability but preserves isolation.

7 Discussion and Limitations

Volatile data remains volatile. TESSERA makes ownership metadata crash-consistent; it does not make pooled DRAM durable. An application requiring data survival must replicate, log, or checkpoint its contents. Systems such as FaRM and RAMCloud already provide such mechanisms [6, 12]; TESSERA can manage their underlying extents but does not replace their recovery protocols.

Availability under long partitions. A client separated from a manager majority can use its current token but cannot renew forever. Increasing token lifetime improves availability and increases the time needed for passive revocation. Explicit generation fencing is not affected, provided the manager can reach the appliance. A partition separating both client and appliance from the majority is intentionally unavailable for reconfiguration.

Hardware trust and multitenancy. The protocol relies on correct gate enforcement and protects against stale or buggy clients, not a compromised appliance. Cryptographic tokens separate tenants, but traffic analysis and shared-fabric side channels remain. A production CXL implementation should integrate generation checks with the fabric manager's isolation domain and audit firmware updates.

Extent granularity. Two-megabyte ownership amortizes metadata and fencing, but internal fragmentation hurts small sparse regions. A hybrid allocator could dedicate packed extents to small allocations while retaining one generation per extent. Our current prototype does not migrate subextent ownership between tenants.

8 Related Work

Disaggregated and far memory. Infiniswap uses RDMA-backed remote swap and decentralized placement [8]; Fastswap studies far memory's effect on cluster job throughput [2]. LegoOS distributes operating-system services across disaggregated hardware [15], while Remote Regions and AIFM expose application-aware interfaces that reduce paging overhead [1, 14]. These systems primarily optimize placement and access. TESSERA is complementary: it specifies how ownership survives control-plane and endpoint failures without scanning the pool. CXL adds coherent pooling and fabric-management mechanisms [5]; generation fencing supplies the failure-atomic handoff that a fabric manager can enforce.

Remote-memory data structures and kernel bypass. FaRM and HERD demonstrate low-latency transactions and key-value access over RDMA [6, 9]. Arrakis removes the kernel from the data plane by giving applications protected hardware resources [13]. Caladan similarly separates a fast data path from a control plane at microsecond timescales [7]. TESSERA adopts this separation, but its unit of protection is a reconfigurable memory extent and its focus is failure recovery rather than request scheduling.

Logs, fencing, and persistent metadata. Shared logs such as CORFU order distributed updates [3], and consensus provides a durable manager order [11]. Chubby popularized sequencers and fencing tokens for distributed locks [4]; TESSERA binds the sequence directly to appliance access control. Frangipani combines distributed locks with a write-ahead log [16]. Persistent-memory systems including Mnemosyne, NOVA, and RECIPE carefully order local persistent updates [10, 17, 18]. TESSERA addresses a different atomicity boundary spanning replicated manager state, volatile compute mappings, and appliance-enforced access.

9 Conclusion

Disaggregated memory should not require choosing between a fast, one-sided data path and predictable failure recovery. TESSERA makes ownership changes failure-atomic with a small durable transition record and a generation checked at the memory appliance. Stable accesses bypass the manager; recovery examines only transitions that were ambiguous at failure time. The prototype's low steady-state overhead

and sub-second recovery across several failure modes suggest a broader systems principle: place consensus around changes in authority, and place an inexpensive monotonic fence where that authority is exercised.

References

[1] Marcos K. Aguilera, Cristiana Amza, Alex Garthwaite, and Andres Lagar-Cavilla. 2018. Remote Regions: A Simple Abstraction for Remote Memory. In *Proceedings of the 2018 USENIX Annual Technical Conference (ATC)*. 775–787.

[2] Emmanuel Amaro, Christopher Branner-Augmon, Zhihong Luo, Amy Ousterhout, Marcos K. Aguilera, Aurojit Panda, Sylvia Ratnasamy, and Scott Shenker. 2020. Can Far Memory Improve Job Throughput?. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys)*. 1–16.

[3] Mahesh Balakrishnan, Dahlia Malkhi, Vijayan Prabhakaran, Ted Wobber, Michael Wei, and John D. Davis. 2012. CORFU: A Shared Log Design for Flash Clusters. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 1–14.

[4] Mike Burrows. 2006. The Chubby Lock Service for Loosely-Coupled Distributed Systems. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 335–350.

[5] Compute Express Link Consortium. 2022. Compute Express Link Specification, Revision 3.0. CXL Consortium.

[6] Aleksandar Dragojević, Dushyanth Narayanan, Miguel Castro, and Orion Hodson. 2014. FaRM: Fast Remote Memory. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 401–414.

[7] Joshua Fried, Zhenyuan Ruan, Amy Ousterhout, and Adam Belay. 2020. Caladan: Mitigating Interference at Microsecond Timescales. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 281–297.

[8] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. 2017. Infiniswap: Efficient Memory Disaggregation with Infiniband and RDMA. In *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 649–667.

[9] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2014. Using RDMA Efficiently for Key-Value Services. In *Proceedings of the ACM Conference on SIGCOMM*. 295–306.

[10] Se Kwon Lee, K Hyun Lim, Hyunsub Song, Beomseok Nam, and Sam H. Noh. 2019. RECIPE: Converting Concurrent DRAM Indexes to Persistent-Memory Indexes. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*. 462–477.

[11] Diego Ongaro and John Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *Proceedings of the 2014 USENIX Annual Technical Conference (ATC)*. 305–319.

[12] John Ousterhout, Arjun Gopalan, Ashish Gupta, Ankita Kejriwal, Collin Lee, Behnam Montazeri, Diego Ongaro, Seo Jin Park, Henry Qin, Mendel Rosenblum, Stephen Rumble, Ryan Stutsman, and Stephen Yang. 2015. The RAMCloud Storage System. *ACM Transactions on Computer Systems* 33, 3 (2015), 7:1–7:55.

[13] Simon Peter, Jialin Li, Irene Zhang, Dan R. K. Ports, Doug Woos, Arvind Krishnamurthy, Thomas Anderson, and Timothy Roscoe. 2014. Arrakis: The Operating System Is the Control Plane. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 1–16.

[14] Zhenyuan Ruan, Malte Schwarzkopf, Marcos K. Aguilera, and Adam Belay. 2020. AIFM: High-Performance, Application-Integrated Far Memory. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 315–332.

[15] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. 2018. LegoOS: A Disseminated, Distributed OS for Hardware Resource Disaggregation. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 69–87.

[16] Chandramohan A. Thekkath, Timothy Mann, and Edward K. Lee. 1997. Frangipani: A Scalable Distributed File System. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles (SOSP)*. 224–237.

[17] Haris Volos, Andres Jaan Tack, and Michael M. Swift. 2011. Mnemosyne: Lightweight Persistent Memory. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 91–104.

[18] Jian Xu and Steven Swanson. 2016. NOVA: A Log-Structured File System for Hybrid Volatile/Non-Volatile Main Memories. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST)*. 323–338.