Adaptive Sharding and Fault Tolerance in Distributed Database Systems

Dissertation

University of Pennsylvania, 2025

Bhavana Mehta

Advisor: Boon Thau Loo, Mohammad Javad Amiri

Committee: Ryan Marcus (chair), Jonathan M Smith, Vincent Liu, Mohammad Javad Amiri (External committee member)

Summary of Changes (Highlights)

- Design a new algorithm to handle multi-item transactions and improve convergence times
- New experimental results to showcase the new algorithm, including YCSB workload
- Code enhancements to fix previous convergence issues – evaluation results are much more stable now
- Marlin accepted to SIGMOD 2026

Outline

- Background
- System Design
- Evaluation
- Related work
- Future Work

Background

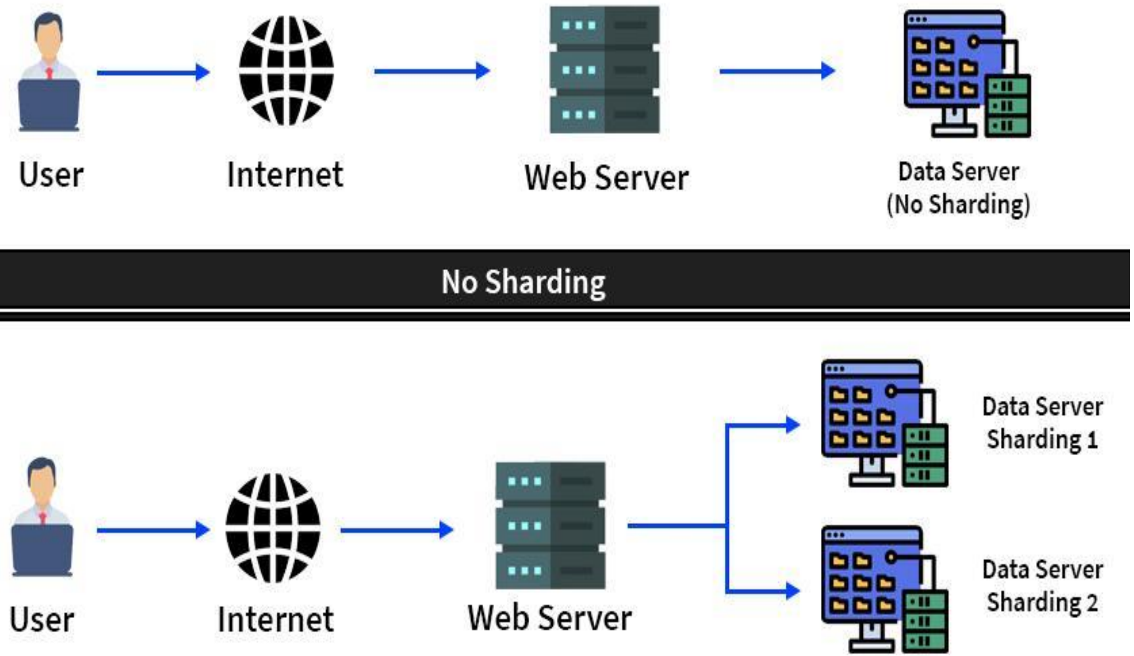
Sharding in Distributed Databases

Distributed Databases:

- Spread data across multiple nodes
- Goal: Scalability, fault tolerance, high throughput

What is Sharding?

- Horizontal partitioning of large datasets
- Each "shard" holds a subset of the overall data



Static vs Dynamic Sharding

Static Sharding:

- Predefined partitions.
- Works well for stable workloads.
- Requires manual intervention for redistribution.

Dynamic Sharding:

- Adjusts shard distribution based on workload.
- Optimizes load balance and reduces cross-shard transactions.

Challenge: Often assumes **trusted** environments

Introducing Adversarial Environments

Trusted vs. Untrusted:

- Nodes can crash vs. nodes can be malicious

Malicious Behaviors:

- Forging data, misrouting transactions, colluding to break consistency

Implication:

- Standard resharding logic may fail if nodes provide false metrics

Fault Tolerant Protocols

Crash Fault Tolerance (CFT):

- Nodes are assumed to fail gracefully.
- Lower latency for cross-shard transactions (O(n)).

Byzantine Fault Tolerance (BFT):

- Nodes can act maliciously.
- Higher latency for cross-shard transactions ($O(n^2)$).
- Requires 3f+1 nodes for fault tolerance.

Problem Statement

Adaptive Sharding:

– Automatically adjust shard boundaries as workloads evolve

Malicious Nodes:

– May falsify load metrics or disrupt data movement

How do we design a robust system that re-shards in adversarial conditions, preserving correctness and performance?

Proposed Solution - Marlin

Marlin:

- Hypergraph partitioning to minimize cross-shard operations
- PBFT for intra-shard consensus
- BFT-2PC for cross-shard transactions

Two architectures based on trust assumptions:

- **Centralized**: Single trusted domain orchestrates rebalancing
- **Decentralized**: Each node can propose updates, validated by quorum

System Design

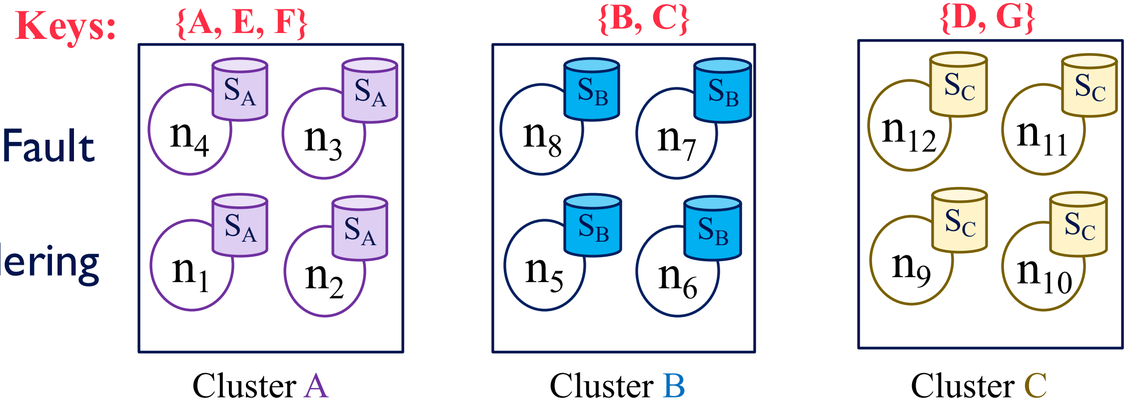
System Design Flow

Monitoring:	Track throughput, latency, malicious indicators
Partitioner:	Hypergraph-based partitioner for minimal data movement
PBFT:	Safeguards shard states (intra-shard)
BFT-2PC:	Atomic cross-shard commits
Re-Configuration:	Either via a central coordinator or distributed proposals

Transaction Execution Model

Intra-Shard Transactions:

- Executed within a single shard.
- Managed using Practical Byzantine Fault Tolerance (PBFT).
- Ensures consistent transaction ordering and fault tolerance within a shard.



Intra-Shard Transactions: **Access key B**

Cross-Shard Transactions: **Access keys A,C**

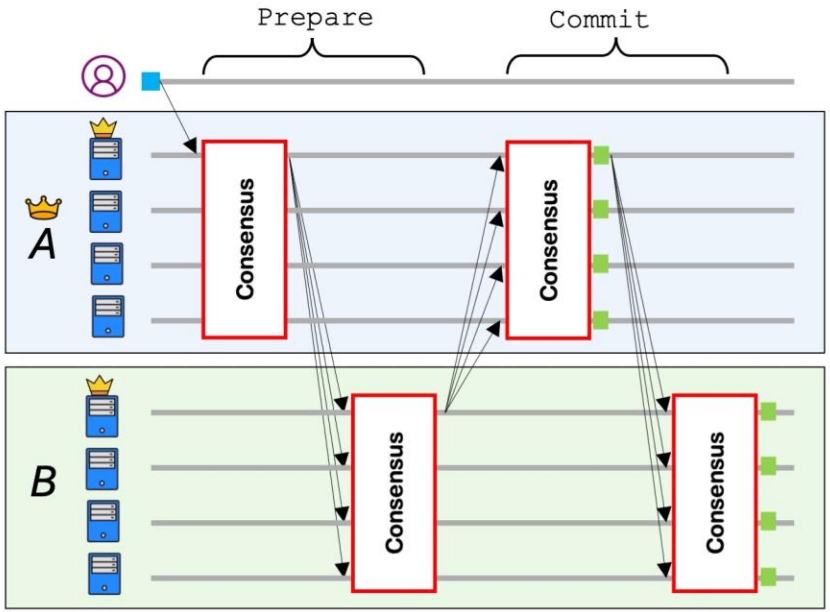
Cross-Shard Transactions:

- Transactions spanning multiple shards.
- Coordinated via Byzantine Fault-Tolerant Two-Phase Commit (BFT-2PC)

Penn Engineering

BFT-2PC: Workflow

- **Initiation**: Coordinator processes the request using PBFT and sends prepare messages.
- **Shard Agreement**: Each shard uses PBFT to agree on the request order and responds with "prepared."
- **Decision**: Coordinator collects responses, runs PBFT, and finalizes commit or abort.
- **Synchronization**: All shards are informed of the final decision.

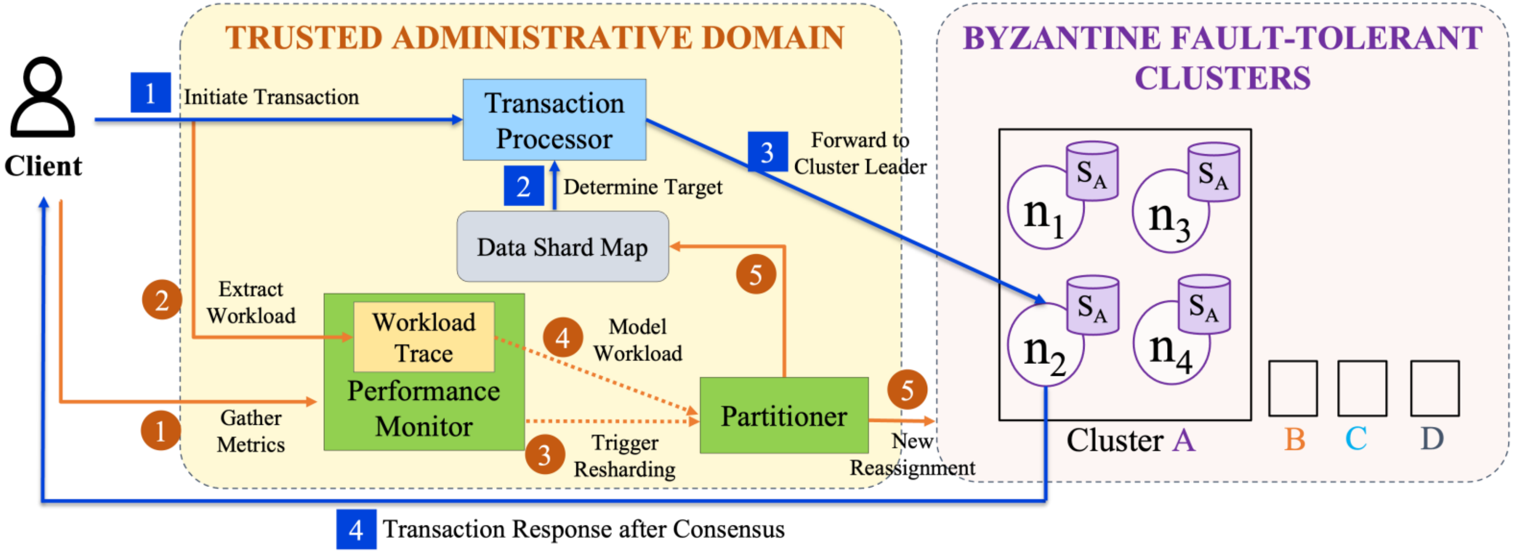


Graph Partitioning Strategy

Hypergraph Partitioning

- Divides a hypergraph $H = (V, E, c, \omega)$ into k balanced blocks.
- Minimizes hyperedges spanning multiple blocks (connectivity).
- Centralized architecture uses KaHyPar, where nodes are grouped into balanced blocks while minimizing inter-block connectivity.
- KaHyPar takes the current shard assignments and transactions, and produces new shard assignments minimizing the cross-shard transactions.

Centralized Architecture



Advantage: Fast, global optimization
Limitation: Single point of failure

Drawbacks of the Centralized Solution

- **Single point of failure** in the central coordinator.
- High **resource overhead** for maintaining administrative domains.
- Susceptible to **targeted adversarial attacks**.

Decentralized Architecture

Objective: Eliminate single coordinator, enhance fault tolerance

Peer-to-Peer: Each node runs a local manager

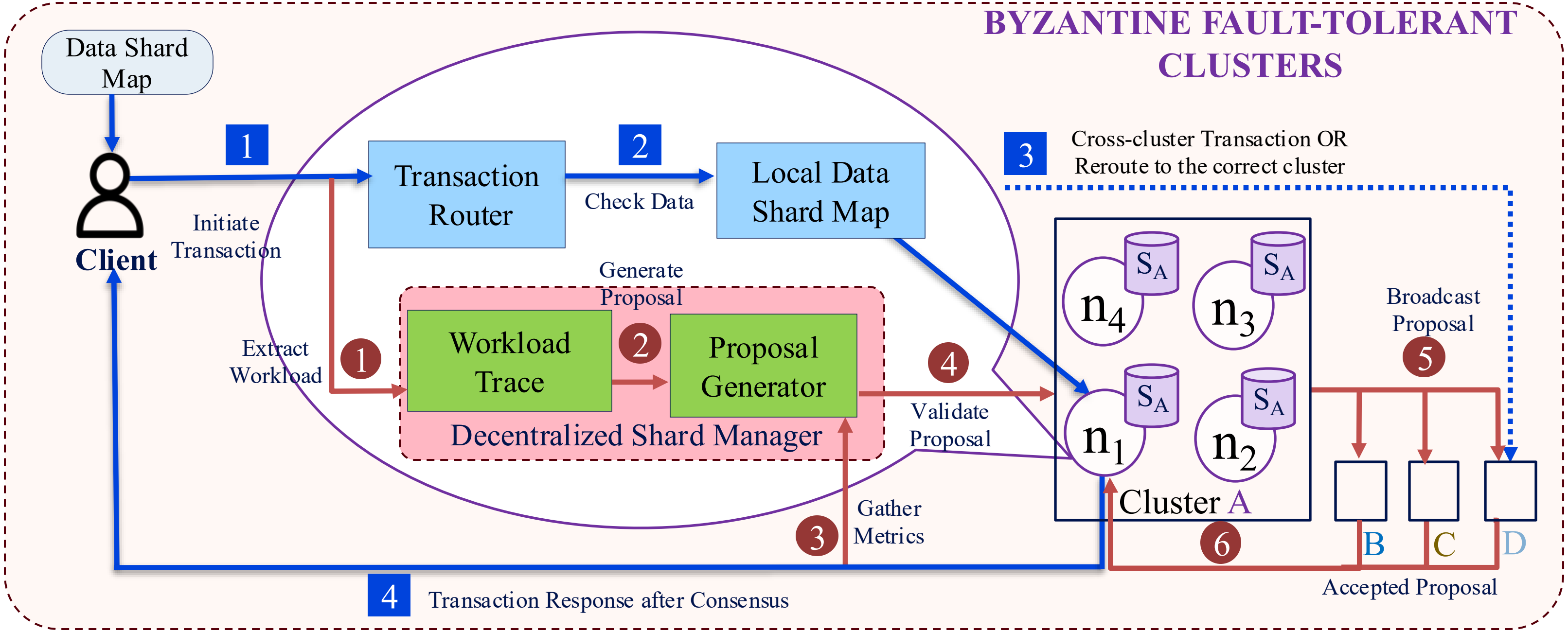
Running Example: Shards : S_1, S_2, S_3; data items {A,B,C,D,E,F}

Overview:

1. Overall Network Diagram
2. Local Data Shard Maps
3. Performance Monitoring
4. Proposal Generation & Validation
5. Cross-shard moves

Penn Engineering

Decentralized Architecture



Performance Monitoring & Local Observations

Per-Shard Metrics:

- Throughput
- Latency
- Cross-shard transactions

Local Logging: Tracks frequent item pairs in cross-shard transactions

Triggered: e.g., 40% drop or 50% cross-shard threshold → consider re-sharding

Proposal Generation: Localized Heuristic Resharding

Objective:

- Co-locate frequently co-accessed items

Localized Heuristic Resharding (LHR) steps:

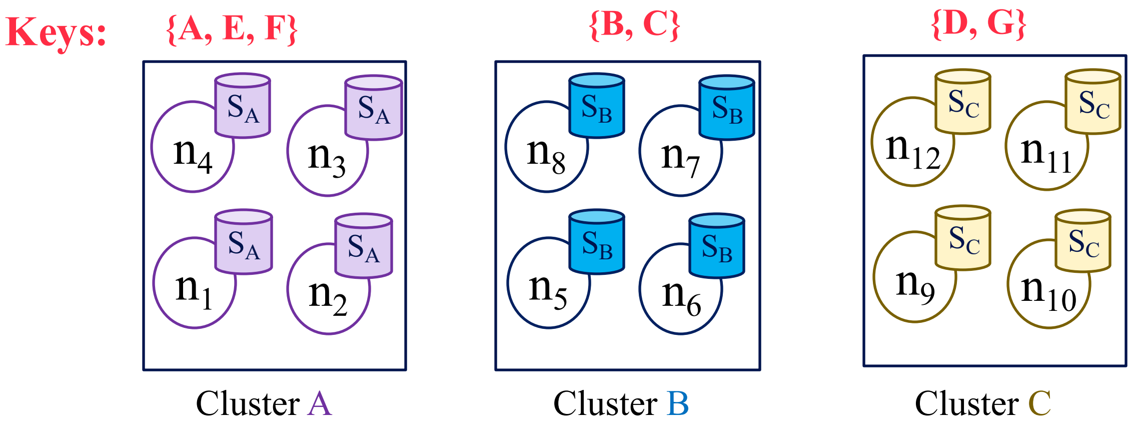
- Transaction Frequency Sorting
- Data Assignment
- Proposal Submission

Key Affinity Algorithm

Transactions

Tx ID	Accessed Keys
T_1	{A, B}
T_2	{C, D}
T_3	{C, D}
T_4	{E}
T_5	{A, B}
T_6	{B, A}
T_7	{E, F}
T_8	{E, G}

Initial Key Distribution



Transaction Frequency Sorting

Accessed Keys	Frequency	Shards Involved
{A, B}	3	S_A, S_B
{E, F}	1	S_A
{E, G}	1	S_A, S_C

Generated Proposal

Move Key A from Cluster A to Cluster B

Proposal Validation

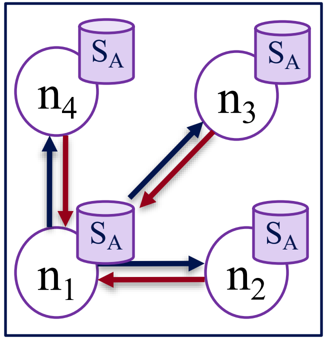
Generated Proposal

Move Key A from Cluster A to Cluster B

Proposed Data Assignment

Shards	Keys
S_A	{E, F}
S_B	{A, B, C}
S_C	{D, G}

Proposal Validation in Cluster A



Cluster A

Proposal Evaluation

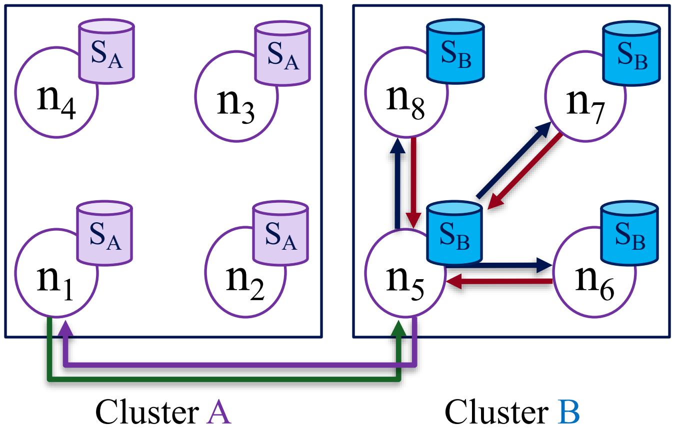
Validated Proposal

Move Key A from Cluster A to Cluster B

Proposed Data Assignment

Shards	Keys
S_A	{E, F}
S_B	{A, B, C}
S_C	{D, G}

Broadcasting Proposal to Other Clusters



Executing the Resharding Plan

- Involved clusters perform BFT-2PC to execute the following:
  - Add Key A to Cluster B
  - Delete Key A from Cluster A
- All nodes update their local Data Shard Maps to reflect the new data distribution

Transactions

Tx ID	Accessed Keys
T_1	{A, B}
T_2	{C, D}
T_3	{C, D}
T_4	{E}
T_5	{A, B}
T_6	{B, A}
T_7	{E, F}
T_8	{E, G}

Finalized Proposal

Move Key A from Cluster A to Cluster B

Final Data Assignment

Shards	Keys
S_A	{E, F}
S_B	{A, B, C}
S_C	{D, G}

Penn Engineering

Evaluation

Experimental Setup

Hardware

- **Machines**: 8 with Intel Xeon Platinum 8253 CPUs (16 cores, 2 threads per core).
- **Memory**: 128 GB DDR4, 1 TB SSD per machine.
- **Network**: 10 Gbps Ethernet.
- **OS**: Ubuntu 20.04 LTS.

Cluster Configuration

- 4 clusters (4 nodes each), total 16 nodes.
- Fault tolerance: f=1 Byzantine fault per cluster.
- Logical processors: 4 per node.

Software

- Implemented in Python.
- Redis-backed shard-to-data mapping for range-partitioning.

Workload

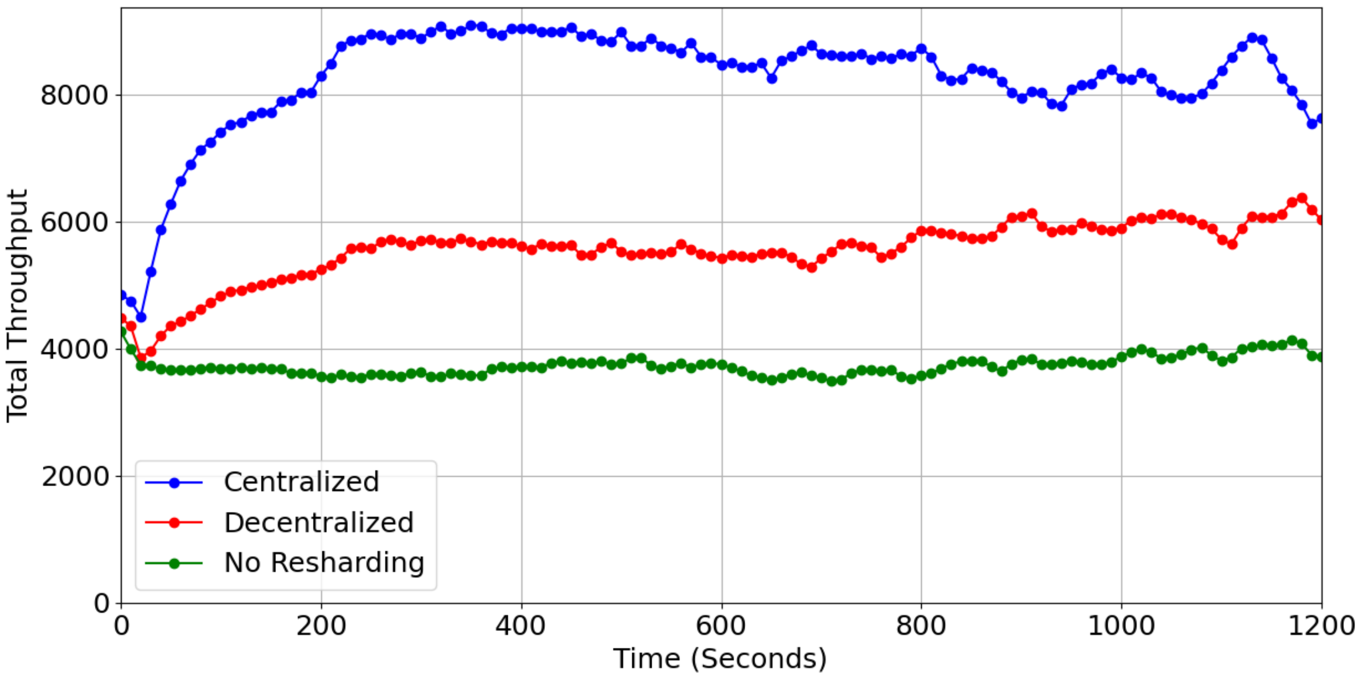
Data and Transactions

- **Data Model**: Key-value store,
- N=10,000 records.
  - Range partitioned across 4 clusters.
  - Full replication within each cluster.
- **Transaction Types**:
  - **Intra-shard**: Single key/item is accessed from a shard
  - **Cross-shard**: Involve multiple shards- two different keys from different shards are involved in one transaction

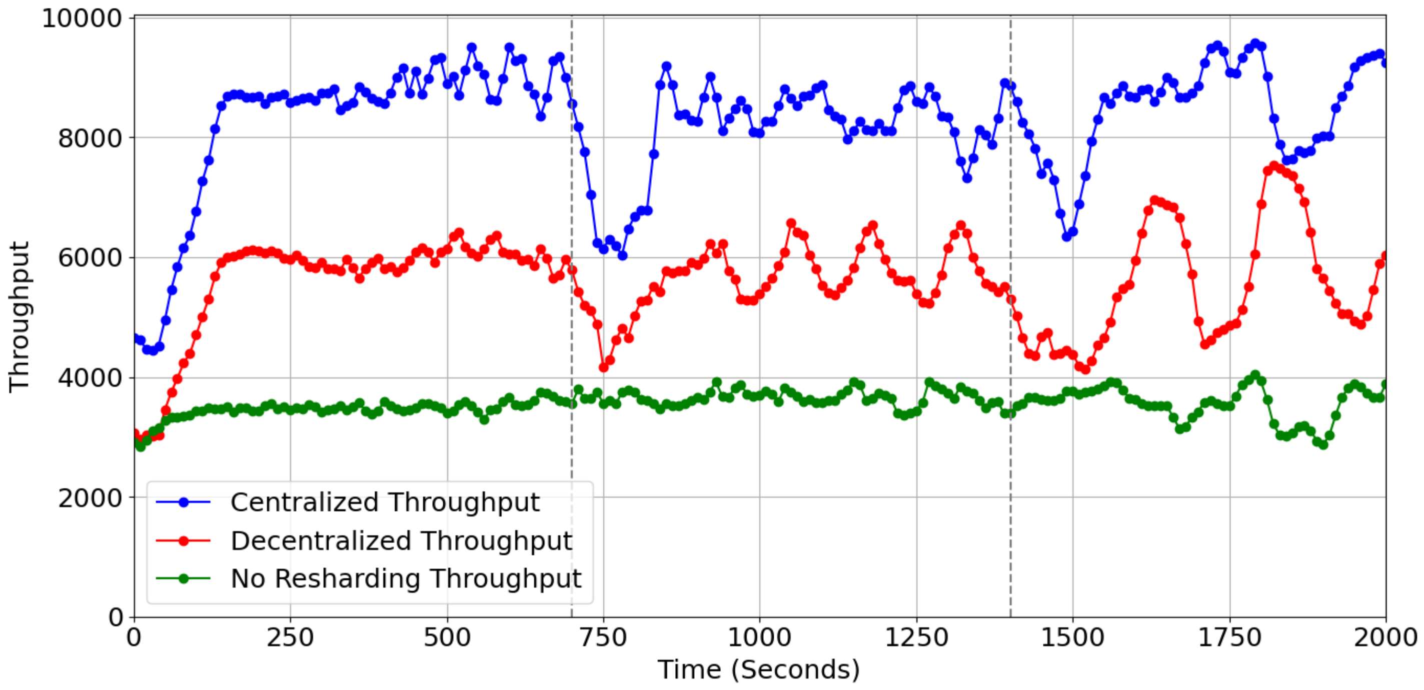
Hotspot and Workload Configurations

- **Hotspot**: 70% of transactions target popular records.
- **Default Transaction Mix**:
  - 40% cross-shard, 60% intra-shard.

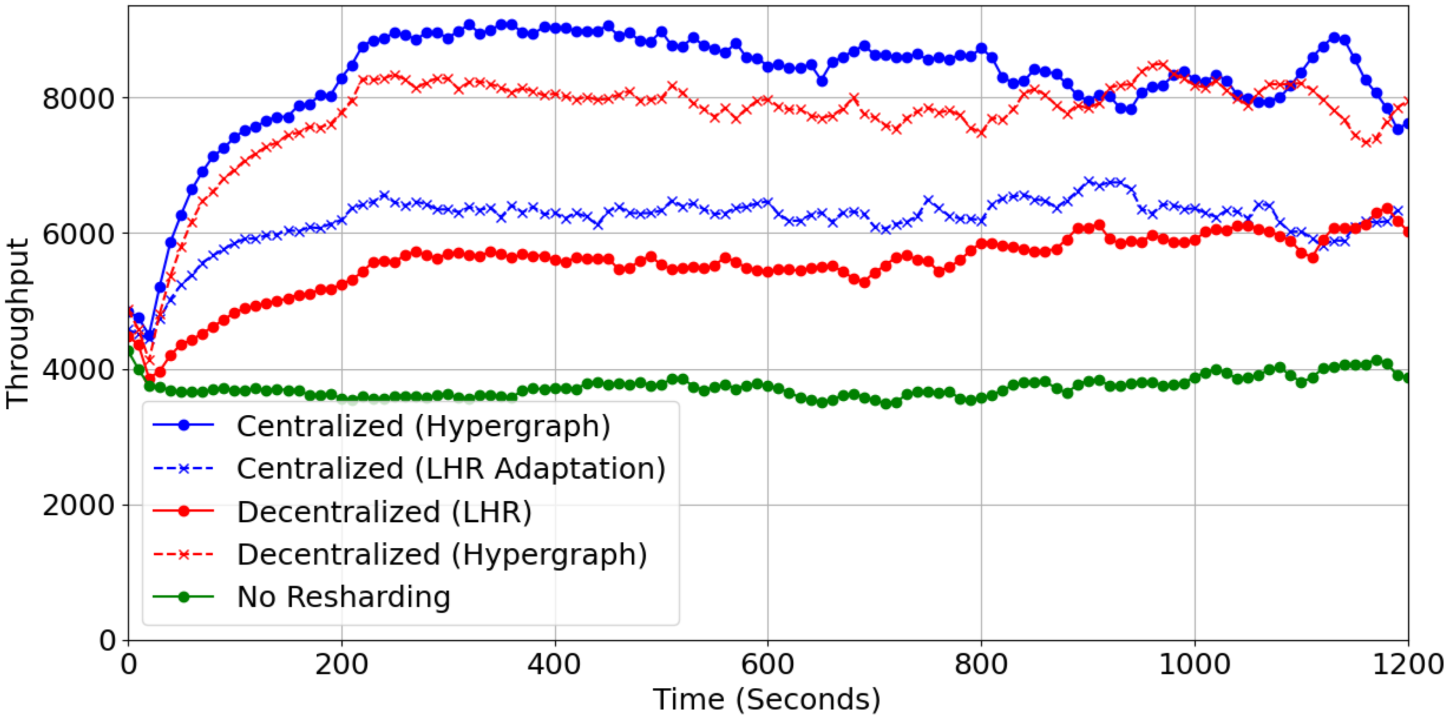
Performance



Adaptability



Scalability



Takeaways

Performance Gains

- Centralized: 2.1x improvement with rapid optimization.
- Decentralized: 1.5x improvement with enhanced fault tolerance.

Adaptability

- Effective resharding maintains throughput across dynamic workloads.
- Centralized: Faster convergence and higher throughput.
- Decentralized: Resilient, eliminates single points of failure.

Partitioning Impact

- Hypergraph partitioning boosts throughput by ~35% compared to heuristic resharding.
- Optimized strategies reduce cross-shard communication.

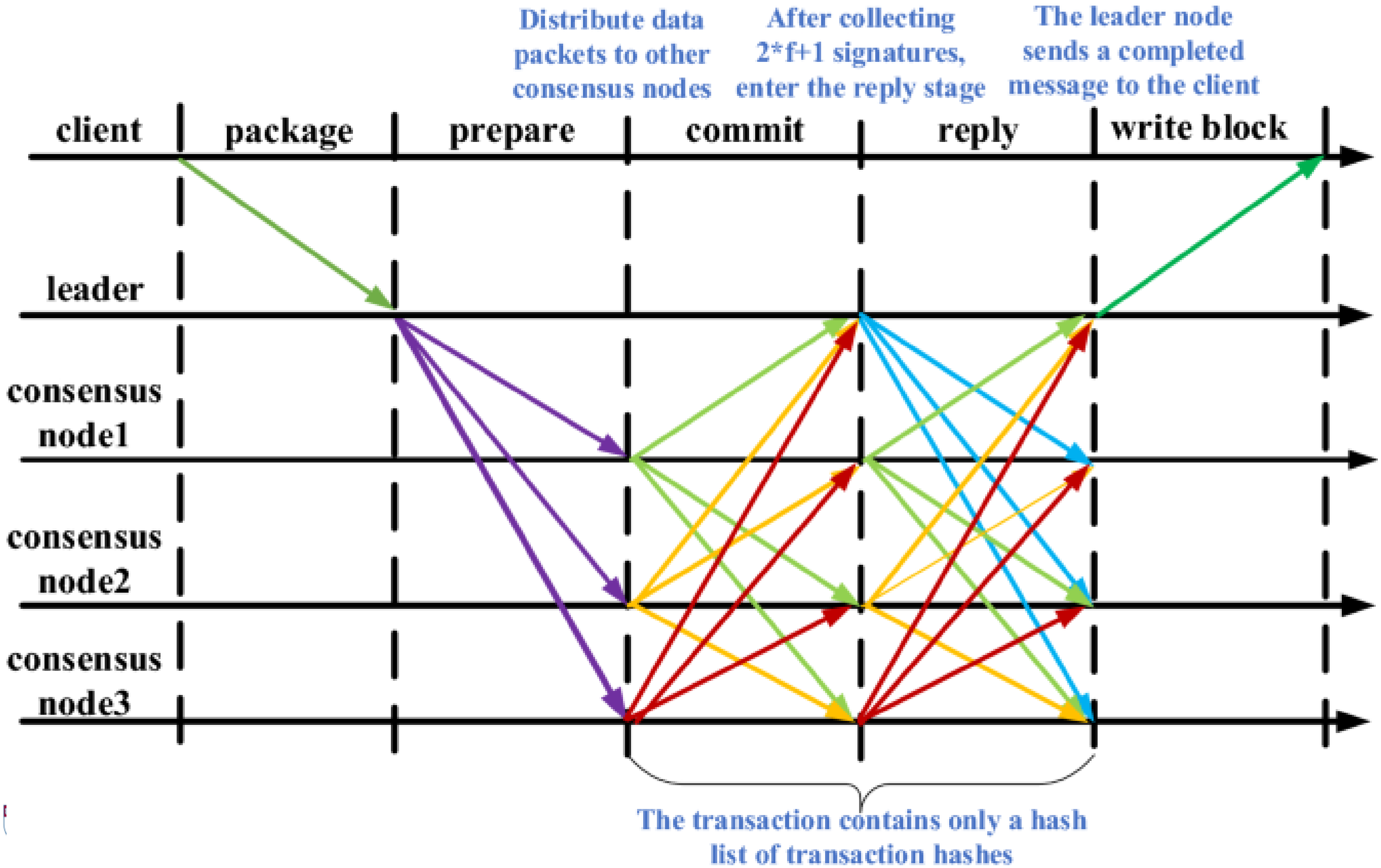
Related Work

1. Byzantine Fault Tolerance (BFT)	•PBFT (Castro & Liskov, 1999): Foundational protocol; high communication cost O(n^2). •SharPer, Blockplane: Address scalability but limited dynamic adaptability.
2. Dynamic Sharding	•SWORD, Schism: Effective in trusted environments; lack fault tolerance. •SharPer: Sharding for untrusted environments but static.
3. Optimization Techniques	•Hypergraph Partitioning: SWORD, Schism for workload balance. •ML-Based Models: Gupta et al. (2021), Haroon et al. (2024).
Challenges in Prior Work	•Limited integration of dynamic adaptability with BFT. •High cost of cross-shard transactions in adversarial settings.
Marlin's Contribution	•Combines adaptive sharding with robust BFT. •Achieves scalability, fault tolerance, and resilience under dynamic, adversarial workloads.

Other Works

- Marlin
  - "Adaptive Sharding in Untrusted Environments"
    VLDB 2025 (Under Submission)
- RLShard
  - "Towards Adaptive Fault-Tolerant Sharded Databases"
    AIDB, VLDB 2023 (Published)
- AdaChain
  - "AdaChain: A Learned Adaptive Blockchain"
    VLDB 2023 (Published)

Thank You!



Distribute data packets to other consensus nodes
After collecting 2*f+1 signatures, enter the reply stage
The leader node sends a completed message to the client
client
package
prepare
commit
reply
write block
leader
consensus node1
consensus node2
consensus node3
The transaction contains only a hash list of transaction hashes

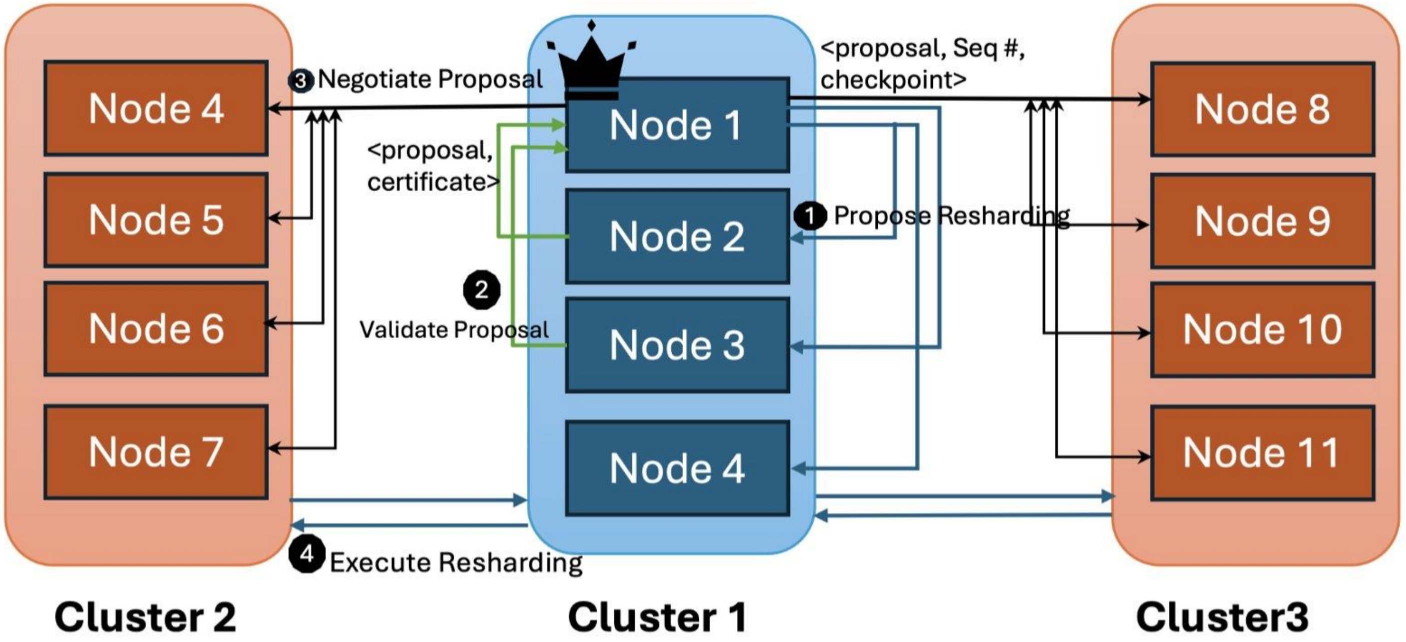
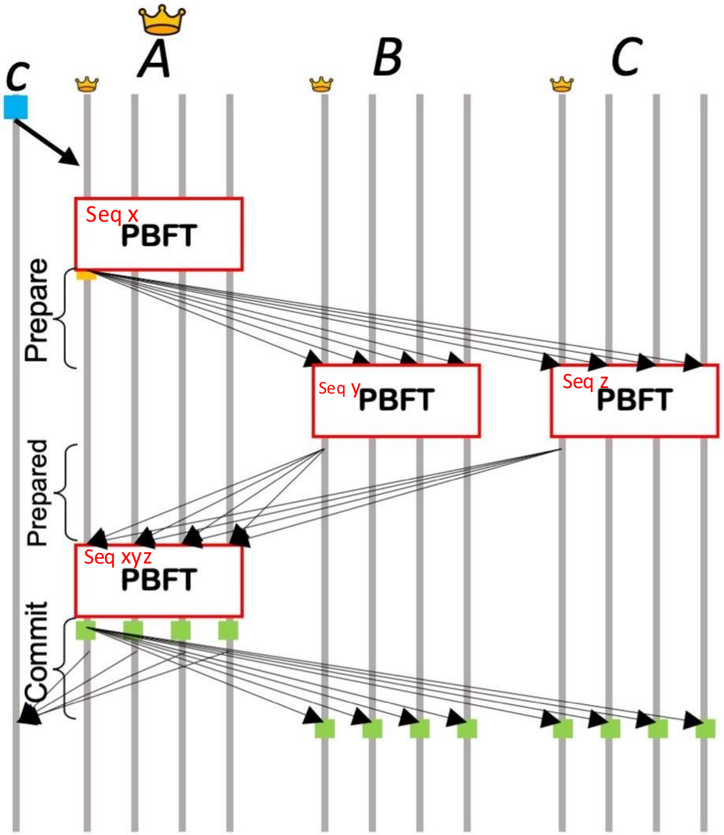


Figure 1: Decentralized Architecture


c
A
B
C
Seq x
PBFT
Prepare
Seq y
PBFT
Seq z
PBFT
Prepared
Seq xyz
PBFT
Commit

Seq y Seq z