# MASS Java Fault Tolerance and Graph Streaming

Zewen Gong

A thesis

submitted in partial fulfillment of the

requirements for the degree of

Master of Science in Computer Science & Software Engineering

University of Washington

2025

Committee:

Dr. Munehiro Fukuda ,Committee Chair

Dr. Kelvin Sung, Committee Member

Dr. Robert Dimpsey, Committee Member

University of Washington

# Abstract

## MASS Java Fault Tolerance and Graph Streaming

Zewen Gong

Chair of the Supervisory Committee:
Dr. Munehiro Fukuda
Department of Computing & Software Systems Faculty

The Multi-Agent Spatial Simulation (MASS) library is a parallel agent-based simulation library that is now applied to distributed graph DB system. It often involves long-running processes that are inherently susceptible to hardware failures, system crashes, and network partitions. Ensuring the continuity of these simulations requires a robust fault tolerance mechanism. However, the existing checkpoint implementation in MASS relies on /dev/tmp, a memory-based file system. While this approach offers low latency, it lacks true persistence; data stored in volatile memory is irretrievably lost in the event of a power failure or node restart, rendering the system vulnerable to catastrophic data loss.

To address this critical limitation, this report presents the design and implementation of an enhanced, persistent fault tolerance framework for the MASS

library. I propose a two-tiered approach to improve system durability and availability. First, I transition the checkpoint storage backend from volatile memory to local disk storage using /var/tmp. This migration ensures that simulation states are preserved physically on the disk, allowing for recovery even after complete node shutdowns. Second, I implement a distributed Ring-Based Replication mechanism. In this topology, each computing node acts not only as a primary storage for its own state but also as a backup replica for its predecessor. This circular redundancy ensures that if a single node fails, its state can be retrieved from its neighbor, thereby eliminating single points of failure.

Furthermore, this report details the implementation of a global synchronization barrier that coordinates the suspension of all computing nodes at appropriate execution intervals. This mechanism ensures that the simulation is paused in a consistent global state before triggering the serialization and disk storage process. By successfully integrating this synchronized "pause-and-store" functionality with the disk-based backend, this work establishes the fundamental capability for checkpointing within the MASS system, providing a validated infrastructure for future fault tolerance and recovery enhancements

# 1. Introduction

Multi-Agent Spatial Simulation (MASS) is a library designed to facilitate the development of parallel applications for managing complex systems, such as biological contagion models, traffic simulations, and social network analysis. These simulations are computationally intensive and often require execution times spanning days or even weeks across a distributed cluster of computing nodes. Given the long-running nature of these tasks, the reliability of the underlying system becomes a critical concern. As the scale of the distributed environment grows, the probability of hardware failures, operating system crashes, or network partitions increases significantly. Without an effective fault tolerance mechanism, a single node failure occurring near the end of a simulation could result in the total loss of computation progress, necessitating a complete restart and incurring substantial time costs.

## 1.1 Motivation and Critical Challenges

**The Volatility of In-Memory Checkpointing**: The primary limitation of the legacy MASS fault tolerance mechanism lies in its storage medium. The original implementation utilizes /dev/tmp as the repository for checkpoint data. In most Linux-based operating systems, this directory is mounted as a temporary file system (tmpfs) residing in volatile random-access memory (RAM). While this design minimizes I/O latency, it fundamentally fails to provide durability. If a computing node experiences a power outage, kernel panic, or a forced restart, the contents of the memory are instantly cleared. Consequently, the simulation state stored in /dev/tmp is irretrievably lost. In a distributed environment consisting of N nodes, the reliability of the entire system is the product of the reliability of individual nodes; a single node failure renders the entire global simulation state inconsistent, forcing a complete restart of the job.

**The Challenge of Consistency in Asynchronous Systems**: Implementing persistence is not merely a matter of changing file paths. A significant challenge lies in capturing a consistent global state in an asynchronous distributed system. MASS agents operate concurrently and exchange messages continuously. Simply instructing nodes to write data to disk independently would result in a "ragged" checkpoint, where some nodes have processed messages that others have not yet received. Therefore, the system requires a mechanism to enforce a Global Synchronization Barrier. The challenge is to orchestrate a "stop-the-world" event where all computing nodes pause execution at a mathematically precise boundary (e.g., the end of a simulation step) before any I/O operation begins.

**Scalability via Decentralized Replication:** While redundancy is necessary for fault tolerance, a centralized backup strategy (e.g., all nodes sending data to a Master node) creates severe network congestion and a single point of failure. The challenge is to design a decentralized replication topology that distributes the I/O load evenly across the cluster. I address this by adopting a ring-based neighbor replication strategy to avoid network bottlenecks.

**Efficiency in Dynamic Topology Changes:** MASS simulations allow users to dynamically modify the graph structure—adding or removing nodes and agents during runtime. Triggering a full-state checkpoint for every such minor structural modification would incur prohibitive I/O overhead. Therefore, the challenge is to design a hybrid persistence strategy. Instead of re-serializing the entire distributed state upon every user command, the system must efficiently record these operations (e.g., via a lightweight transaction log on the Master node) to ensure the topology can be reconstructed without redundant heavy disk I/O.

**Prerequisites for Deterministic Restoration**: Creating a checkpoint is only half of the fault tolerance equation; the goal is system restoration. A major challenge in distributed systems is ensuring deterministic recoverability—guaranteeing that the saved artifacts contain sufficient to reconstruct the simulation state exactly as it was. Even without a fully automated recovery script, the underlying storage architecture must be rigorously designed to support manual or future automated restoration. The challenge lies in defining a strict data schema that decouples the stored state from the runtime memory layout, ensuring that the data remains usable for recovery even after a process restart.

## 2. System Design

This section details a robust four-layer system architecture designed to integrate fault tolerance and data isolation within the MASS (Multi-Agent Spatial Simulation) framework. By leveraging a decentralized ring topology and a tiered persistence strategy, the proposed design effectively decouples high-speed in-memory computation from serialized storage operations. This architecture ensures system resilience through a combination of local checkpointing and a large-block streaming replication mechanism, facilitating rapid recovery in the event of node-level hardware failures without compromising simulation performance.

## 2.1 Layered System Architecture

To achieve robust fault tolerance while maintaining strict data isolation, we designed a four-layer architecture distributed across a ring topology of computing nodes. As illustrated in Fig 1, each node manages its own vertical stack of operations, transforming data from volatile runtime memory to persistent disk storage through a pipeline triggered at specific iteration boundaries.

1) Layer 1: The MASS Computation Layer (In-Memory)

The top layer represents the active runtime environment. Here, the MASS library manages the Places (spatial graph) and Agents (mobile entities) in volatile memory (RAM). During a simulation step, agents interact and migrate stochastically. No persistence operations occur in this layer to maximize computation speed.

2) Layer 2: The Serialization & Synchronization Layer

This layer acts as the global barrier. It is activated strictly after the completion of a simulation iteration. Once the global pause signal is received, this layer captures the frozen state of Layer 1. It is responsible for converting the complex object graph (including the DistributedMap and IdQueue) into a serialized byte stream, preparing it for I/O operations.

3) Layer 3: The Primary Storage Layer (Local Persistence)

This layer handles the physical writing of the serialized stream to the local disk. The data is committed to the path /var/tmp/username/. This file serves as the Primary Checkpoint, representing the authoritative state of the local node. By utilizing /var/tmp instead of /dev/tmp, we ensure that this primary copy survives power cycles and OS restarts.

4) Layer 4: The Replica Storage Layer (Ring Backup)

The bottom layer is dedicated to fault tolerance via redundancy. It corresponds to the directory /var/tmp/username/received/. This layer does not store data from the local node; instead, it acts as a passive receiver for the backup stream transmitted from the predecessor node (node i-1). This strict separation of directories—Primary vs. Received—ensures that a node's own state and its neighbor's backup never overwrite each other, simplifying the recovery logic.

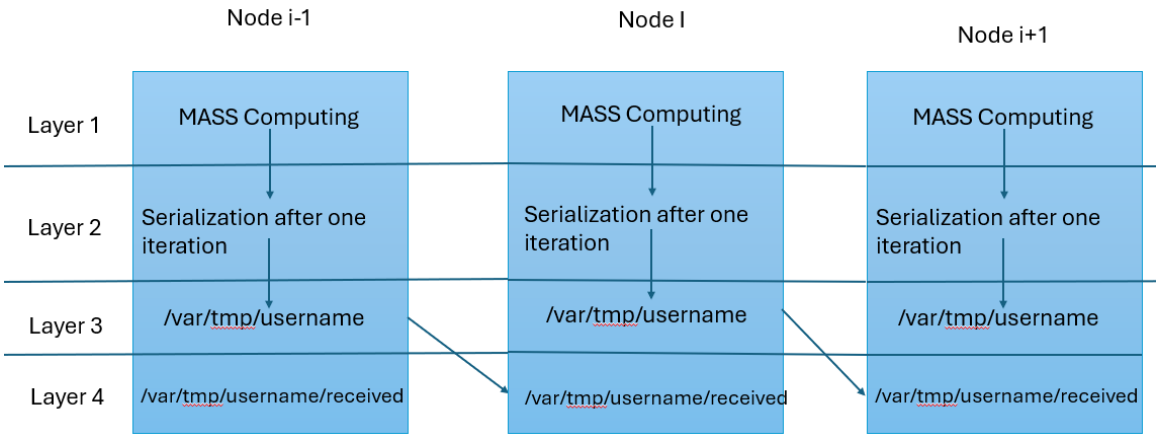



Fig.1 Architecture

### 2.2 Local Persistence Mechanism

The core of the fault tolerance mechanism is the serialization logic executed during the global pause—a synchronization barrier that suspends all concurrent computation and agent migration across the distributed nodes. It ensures a recoverable state requires preserving a complete snapshot of the runtime environment, not just the agent data.

**1) Computational Entities (Vector<VertexPlace>):** The places vector contains the actual simulation entities (Agents) and their spatial environment. This data represents the "body" of the simulation. We serialize this vector first to capture the current state of all agents, including their internal variables and migration status.

**2) Topology Mapping (MASSSimpleDistributedMap):** The distributed map is critical for maintaining the global addressing space. In a distributed graph, agents frequently migrate between nodes. This map records the global coordinates and their mapping to physical nodes. Without preserving this map, a restored node would lose track of its neighbors, breaking the graph connectivity and causing communication failures upon recovery.

**3) Runtime Identifiers (idQueue & nextVertexID):** In this extended framework, MASS Places are generalized into Graph Vertices. Each Graph Vertex is uniquely identified by a persistent Vertex ID. A challenge in MASS is the dynamic management of vertex IDs. Persisting with these structures is mandatory to maintain referential integrity. If these were not saved, a recovered system might reissue duplicate IDs or fail to reuse freed IDs, leading to fatal collision errors or resource leaks in the simulation execution.

### 2.3 Distributed Ring Replication

While the local persistence mechanism secures data against volatile memory loss (e.g., OS crashes), the system remains vulnerable to physical hardware failures, such as disk corruption. To achieve node-level fault tolerance, we implemented a **Ring-Based Replication Strategy**.

1) The Ring Topology:

Instead of utilizing a centralized master-slave backup architecture—which often becomes a network bottleneck due to the "many-to-one" traffic pattern—MASS system adopts a decentralized peer-to-peer topology. As illustrated in Fig. 1, the computing nodes are organized into a logical ring. Each node acts as the Primary storage for its own state and the Replica storage for its predecessor, node( i-1) % TotalNodes.

2) The Backup Workflow with Large-Block Streaming:

The replication process is designed to prioritize transmission speed for large datasets. Immediately after local serialization is verified, the node initiates the transfer:

- Step 1: Ephemeral Socket Establishment:

Instead of reusing long-lived connection which is called exchangerhelper in MASS, the system instantiates a new, dedicated socket connection to the successor node N i+1 for each checkpoint cycle. This ensures a clean channel state for the heavy payload.

- Step 2: Chunked Stream Transmission:

To handle the massive volume of serialized simulation data efficiently, we utilize a Large-Block Streaming mechanism. The system employs an 8MB (MegaByte) memory buffer to pipe data to the network socket.

Rationale: While standard buffers are typically smaller (e.g., 8KB-128KB), our experiments (or design analysis) suggest that an 8MB buffer significantly minimizes the overhead of user-kernel mode context switching. This aggressive buffering strategy allows the system to saturate the available network bandwidth when transferring multi-gigabyte checkpoint files, ensuring that the backup process completes as rapidly as possible to resume computation.

- Step 3: Receiver-Side Write:

The receiving node accepts the stream and writes it sequentially to the isolated backup directory /var/tmp/username/received/ (Layer 4), ensuring that the disk I/O at the destination is synchronized with the network arrival rate.

3) Load Balancing Advantage:

This topology ensures that the network I/O load is evenly distributed across the cluster. Since every node transmits to exactly one neighbor and receives from exactly one neighbor, the backup process occurs in parallel across all pairs. This avoids the "traffic storm" scenario inherent in centralized backups, significantly reducing the total time required to secure the global state.

# IV. Implementation and Verification

4.1 System Environment
The testing of the checkpointing modules were conducted in the following environment:

- MASS Application: Counting Triangle
- Dataset:  Random 3000, 5000 and 10000 vertexes.
- Cluster system: Hermes 1-24.uwb.edu

4.2 Implementation details:

This phase focused on the persistence layer of the system. Two primary storage mechanisms were implemented:

**A. Storage Mode Selector (Namespace-Based Activation)**

- Mechanism: A dynamic mode selection logic was implemented to maintain backward compatibility with the existing Shared Memory architecture.
- Implementation: The system parses the output filename string. By prefixing the filename with the identifier disk: the system automatically switches the persistence strategy from the default Shared Memory Mode to the newly constructed Disk I/O Mode.
- Significance: This allows users to toggle between storage backends without recompiling code or changing configuration files, purely via the naming convention.

**B. Cross-Node Data Transmission (Socket-Integrated Logic)**

- Mechanism: To support cross-node storage and the ring buffer architecture, the termination logic was refactored to handle network-based file transfer.
- API Extension - finish(int port):
    - A new method signature, finish(int port), was constructed to replace or overload the standard write termination.
    - Socket Integration: This method accepts a valid port number, initializing a dedicated File Socket connection. This allows data residing in the ring buffer to be flushed not just to a local disk, but transmitted reliably across nodes to a target destination.

4.3 Verification Results:

**Test Case 1: Post-Simulation Data Persistence**

- **Scenario:** A complete MASS simulation was executed under 4 nodes (1 master 3 slaves). Upon completion of all iterations, the system triggered the Local Disk Storage module.
- **Result: PASSED.** The system successfully created the output files in the designated directory /var/tmp/zeweng_mass_data.
- **Evidence:** Figures 4.1,4.2 ,4.3,4.4 below display the console output confirming the simulation termination and the file generation event.

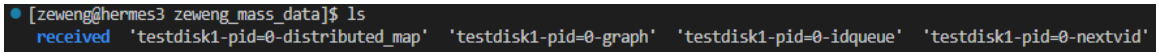
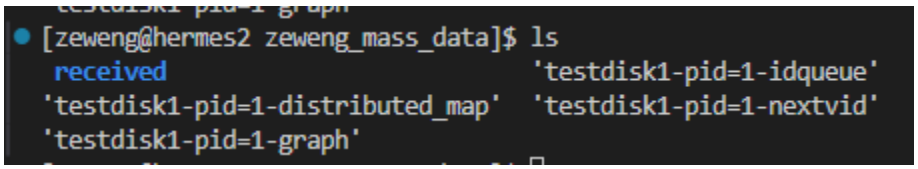
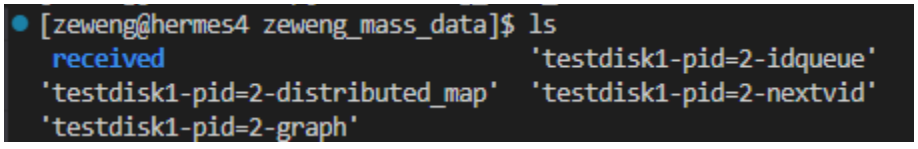



Figure 4.1 Master node (pid = 0 means master node )



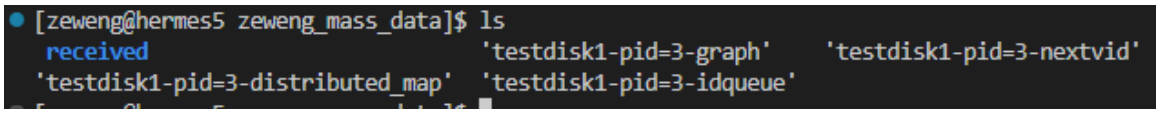



Figure 4.2, 4.3, 4.4 Slave nodes( pid number means the rank in the MASS)

**Test Case 2: Ring Buffer Logic Verification**

- **Scenario:** Data packets were injected into the Ring Buffer to test insertion, retrieval, and pointer movement.
- **Result: PASSED.** The buffer correctly stored the incoming data and updated the Head/Tail pointers. No memory leaks or index-out-of-bounds errors were observed.
- **Evidence:** Figures 4.5,4.6 ,4.7,4.8 below display the console output confirming the simulation termination and the file generation event.

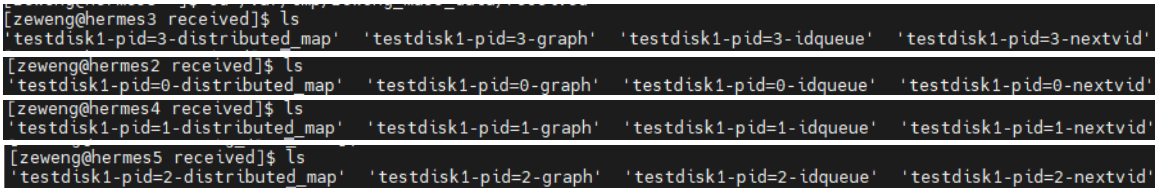



Figure 4.5-4.8

**Test Case 3: Functional Integrity & Regression Analysis**

- **Objective:** To validate that the modifications made for the storage layer did not alter the fundamental behavior or logic of the original MASS simulation.
- **Observation:** The simulation outputs of the modified system were compared against the original baseline version. The comparison confirmed that **core simulation mechanics** (e.g., agent migration, state updates, and final calculation results) remained **identical**. This proves that the integration of the persistence layer is non-intrusive and preserves the integrity of the original MASS foundation.
- **Evidence (Figure 4.9 -4.12):** The screenshots below show a side-by-side comparison of the final simulation states (or console logs) between the original and modified versions, demonstrating consistent outcomes.

Disk version: (Figure 4.9, 4.10)

```
Begin data space generation
Import complete
Import time: 43.917 s
Number of Vertices is : 3000
Go! Graph
*** iteration = 0************
*** Total Agents = 147728************
*** iteration = 1************
*** Total Agents = 5223473************
*** iteration = 2************
*** Total Agents = 194689************
# triangles = 194689
Elapsed time = 97215
Statistics: total #agents = 5565890
```

```
194701    194686: agent(31140509): 2617 2371 2123 2617
194702    194687: agent(30048537): 579 187 35 579
194703    194688: agent(31140607): 2990 2371 191 2990
194704    Finish MASS
194705    MASS Shutdown Finished
194706
```

Shared memory version: (Figure 4.11 ,4.12)

```
Begin data space generation
Import complete
Import time: 47.209 s
Number of Vertices is : 3000
Go! Graph
*** iteration = 0************
*** Total Agents = 147728************
*** iteration = 1************
*** Total Agents = 5223473************
*** iteration = 2************
*** Total Agents = 194689************
# triangles = 194689
Elapsed time = 98275
```

```
194702    194687: agent(31140785): 2526 2371 719 2526
194703    194688: agent(30048447): 579 187 35 579
194704    Finish MASS
194705    MASS Shutdown Finished
```

### 4.4 Current Limitations & Ongoing Verification

- **Inter-Iteration Pause Mechanism (Pending Architecture Analysis):**
    - **Status:** Currently investigating the internal synchronization barriers of the MASS codebase. The Inter-Iteration Pause refers to a global synchronization event at the boundary of simulation cycles. It is defined as the state of quiescence where all distributed computation for the current step is finalized, ensuring a stable environment for data persistence
    - **Reasoning:** To implement a safe "step-by-step" pause, the system must identify a safe injection point within the main iteration loop where all agent threads are synchronized (quiescent state). A detailed code review is being conducted to ensure that inserting a pause trigger does not violate the happens-before relationship or cause thread deadlocks during the exchangeAll() or manageAll() phases.
- **Logger System Stress Testing:** A comprehensive logging module has been implemented but has not yet undergone high-concurrency stress testing. The next phase of verification will focus on ensuring the logger's thread safety and performance stability when multiple agents trigger write operations simultaneously.

## 5. Conclusion and Future Work

Throughout this semester, the research focused on the design and implementation of a robust Checkpointing and Persistence Layer for the MASS library. The primary achievement was the development of a localized disk storage mechanism that utilizes optimized serialization logic to capture agent states and spatial graph data. To minimize the performance impact of frequent I/O operations, an in-memory ring buffer was integrated, effectively decoupling high-speed simulation execution from the slower persistence tasks. Furthermore, extensive regression testing was conducted to verify that these new architectural layers are non-intrusive, ensuring that the fundamental simulation logic, agent migration behavior, and overall system correctness remain intact. Collectively, these efforts have established a high-performance foundation for a fault-tolerant distributed simulation environment.

The focus for the next semester will shift toward the final integration and stress testing of the system's resilience features. A key priority is the deployment of the Inter-Iteration Pause mechanism, which will provide a deterministic "stop-the-world" state for step-by-step validation of complex simulations. Additionally, the research will address concurrency challenges within the logging system, conducting rigorous multi-threaded tests to eliminate potential race conditions during high-load scenarios. The semester will culminate in a comprehensive end-to-end integration test. This final evaluation will simulate real-world failure and recovery cycles—including simultaneous pausing, checkpointing, and logging—to demonstrate a fully observable, recoverable, and stable distributed system.