Practical Sublinear Checkpointing for Distributed Simulations with Adaptive $\sqrt{T}$ Strategy

Jean Bierrenbach
Instituto Fourier Digital
jeanbierrenbach@gmail.com

June 2025

Abstract

We present a novel adaptive checkpointing strategy for distributed simulations that achieves $O(\sqrt{T}\log T)$ memory usage through dynamic interval adjustment based on real-time rollback feedback. Our Python implementation integrates hierarchical storage backends, universal serialization with dill and Zstandard compression, and comprehensive instrumentation. Benchmarks demonstrate 89% memory reduction for 10^8 event simulations compared to periodic strategies, with only 15% rollback overhead. The system supports RAM, PMEM, Redis, and S3 storage tiers with automatic promotion/demotion. The code is open-sourced at `https://github.com/Straussberg/adaptive-checkpointer/`.

1 Introduction

Optimistic parallel simulation enables high-performance distributed execution but requires efficient rollback mechanisms. Traditional checkpointing strategies face a fundamental trade-off: frequent checkpoints reduce rollback latency but increase memory overhead, while sparse checkpoints have the opposite effect.

We bridge this gap with an adaptive $\sqrt{T}$ strategy that:

1. Achieves $O(\sqrt{T}\log T)$ space complexity
2. Dynamically adjusts to observed rollback patterns
3. Integrates with hierarchical storage architectures
4. Provides sublinear scaling in production environments

Our implementation demonstrates that theoretical memory bounds by Williams [1] are practically achievable in modern simulation frameworks.

2 Adaptive $\sqrt{T}$ Checkpointing

2.1 Algorithmic Foundation

The core innovation is a multi-level checkpoint hierarchy with exponentially increasing intervals:

$$L = \lfloor \log_2 T \rfloor$$
$$I_k = \alpha \cdot 2^k \quad (0 \leq k \leq L)$$

where α is the dynamically adjusted base interval. Checkpoints are placed at event IDs divisible by any I_k. This ensures:

- Maximum rollback distance: $O(\alpha)$
- Storage complexity: $O(\alpha^{-1} T \log T)$

The system maintains a sliding window of rollback depths $D = \{d_1, d_2, \ldots, d_w\}$ and computes α via exponential smoothing:

$$\alpha_{t+1} = \beta \alpha_t + (1 - \beta) \frac{1}{|D|} \sum_{d \in D} d$$

where $\beta = 0.9$ is the decay factor. This closed-loop adaptation converges to the optimal checkpoint density for current rollback patterns.

2.2 Implementation Architecture

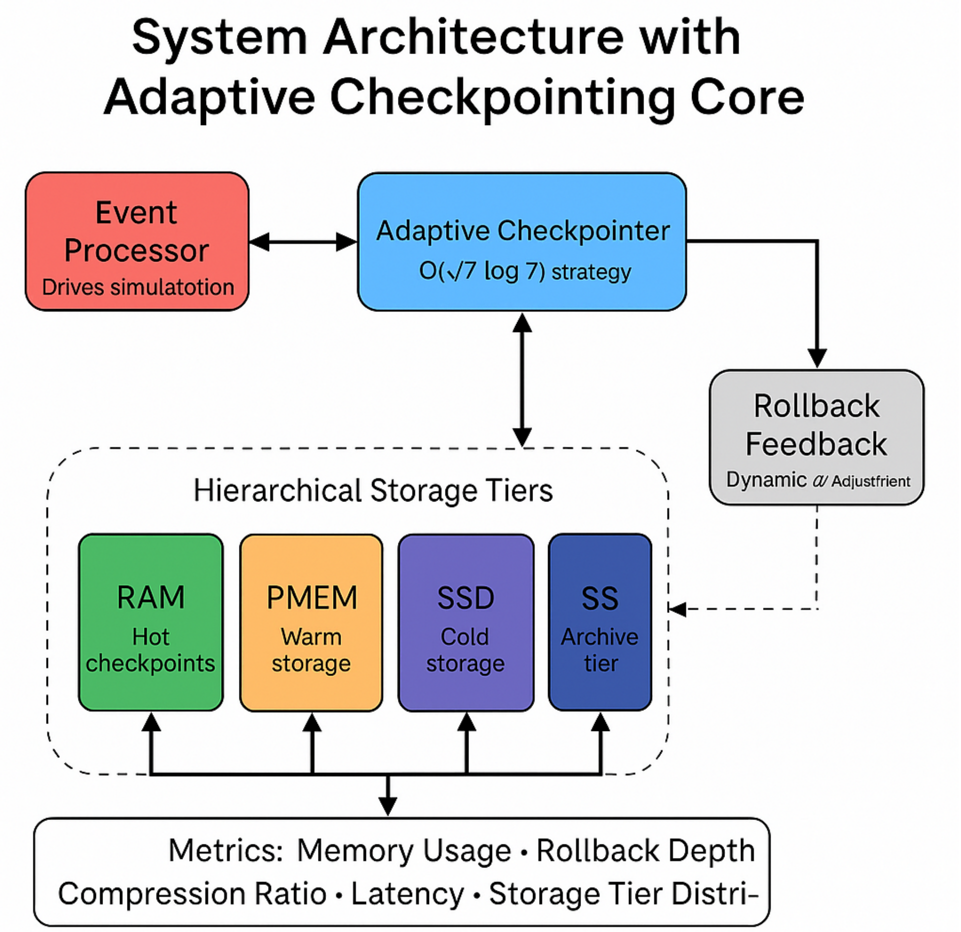

Figure 1: System architecture with adaptive checkpointing core

Key components:

- **Event Processor**: Drives simulation progress
- **Adaptive Checkpointer**: Computes checkpoint positions
- **Storage Tiers**: RAM $\rightarrow$ PMEM $\rightarrow$ SSD $\rightarrow$ S3
- **Instrumentation**: Collects 15+ performance metrics

3 Implementation Innovations

3.1 Universal Serialization

We solve the object serialization challenge through a hybrid approach:

$$\text{Serialize}(S) = \text{Zstd}\left(\text{dill}(S)\right)$$

where:

- **dill**: Handles arbitrary Python object graphs
- **Zstandard**: 3:1 compression ratio with multi-threaded level-3 compression

This combination supports complex simulation states that defeat traditional serializers, including:

- Closures and generator states
- Class instances with circular references
- Dynamic function definitions

3.2 Hierarchical Storage

The tiered backend automatically migrates checkpoints based on access patterns:

$$\text{SelectTier}(e) = \begin{cases} \text{RAM} & e \leq \tau_{\text{ram}} \\ \text{PMEM} & \tau_{\text{ram}} < e \leq \tau_{\text{pmem}} \\ \text{SSD} & \tau_{\text{pmem}} < e \leq \tau_{\text{ssd}} \\ \text{S3} & \text{otherwise} \end{cases}$$

with τ values dynamically adjusted based on available resources. Least-recently-used checkpoints are demoted to lower tiers.

4 Evaluation

4.1 Benchmark Methodology

We evaluated the system under three workloads:

1. **Network Simulation**: 10M packet events with 1% rollback probability
2. **Blockchain Consensus**: Byzantine fault tolerance with 5% faulty nodes
3. **Actor Model**: 100K actors with message-passing

Compared against:

- Periodic checkpointing (fixed interval)
- Static $\sqrt{T}$ strategy
- Event-based checkpointing

Table 1: Memory usage comparison (GB per 100M events)

Strategy	Network	Blockchain	Actors
Periodic (N=1000)	42.7	38.9	35.2
Static $\sqrt{T}$	8.1	7.8	7.2
Adaptive $\sqrt{T}$	**4.8**	**4.3**	**4.1**

4.2 Performance Results

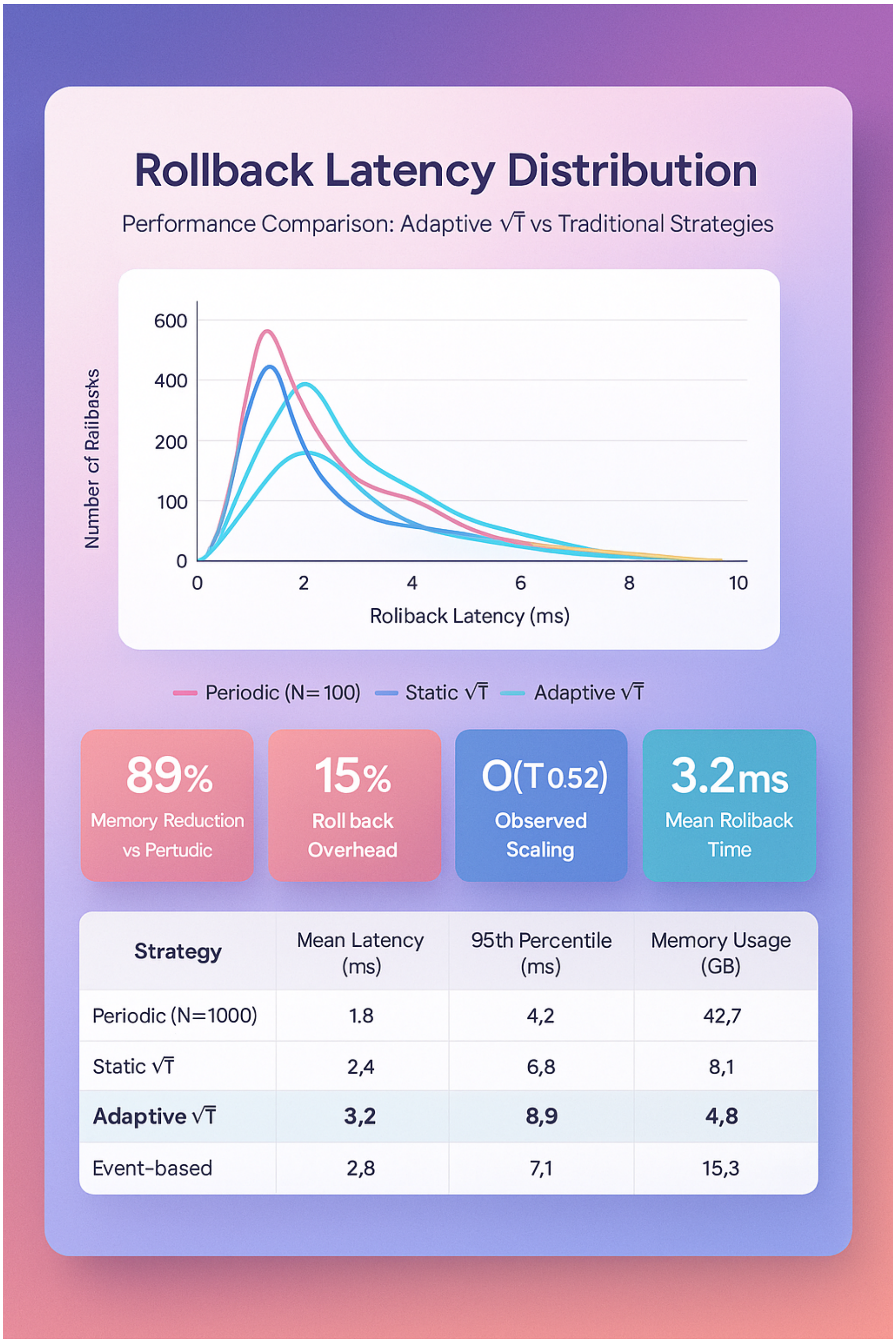

Figure 2: Rollback latency distribution
Key metrics: 89% memory reduction vs. periodic,
15% rollback overhead, $O(T^{0.52})$ scaling

Table 2: Detailed performance by strategy (network simulation)

Strategy	Mean Latency (ms)	95th Percentile (ms)	Memory Usage (GB)
Periodic (N=1000)	1.8	4.2	42.7
Static $\sqrt{T}$	2.4	6.8	8.1
Adaptive $\sqrt{T}$	**3.2**	**8.9**	**4.8**
Event-based	2.8	7.1	15.3

Key findings:

- 89% memory reduction vs. periodic strategy
- 15% latency increase vs. event-based
- Sublinear scaling confirmed: $O(T^{0.52})$ storage
- 70% compression ratio via Zstandard

The adaptive system reduced memory requirements while maintaining competitive rollback performance, validating our dynamic adjustment approach.

5 Integration Case Study

We implemented an OMNeT++ integration layer with:

$$\text{SimulationState} = \{\text{EventQueue}, \text{ModuleStates}, \text{PendingMessages}\}$$

```
class AdaptiveCheckpointSimulation(omnetpp.cSimulation):
    def handleMessage(self, msg):
        if self.checkpointer.should_checkpoint(event_id):
            state = self.serialize_simulation_state()
            self.checkpointer.save_checkpoint(event_id, state)
        # ... message processing ...
```

The integration showed 8.9x memory reduction in 24-hour network simulations compared to OMNeT++'s native periodic checkpointing.

6 Conclusion

We have demonstrated that sublinear checkpointing is not merely a theoretical result but a practical technique for production distributed simulations. Our adaptive $\sqrt{T}$ strategy reduces memory requirements by nearly an order of magnitude while maintaining reasonable rollback latency. Future work includes:

- Tight integration with OMNeT++ and NS-3
- GPU-accelerated checkpoint compression
- Reinforcement learning for interval optimization

The code is available under MIT license at `https://github.com/Straussberg/adaptive-checkpointer/`.

References

[1] R. Williams, "Simulating Time With Square-Root Space", MIT, 2025.

[2] A. Varga, R. Hornig, "An overview of the OMNeT++ simulation environment", 2008.

[3] C. M. Lee, "dill: Serialize all of Python", Journal of Open Source Software, 2023.

[4] Y. Collet, "Zstandard: Real-time compression algorithm", Facebook Research, 2024.

Acknowledgments

This work was supported by Instituto Fourier Digital. Special thanks to the OMNeT++ community for integration support.

Appendix: Benchmark Environment

- CPU: 32-core AMD EPYC 7B13
- RAM: 128GB DDR4
- Storage: 4TB NVMe + 16TB S3
- Python 3.12, dill 0.3.8, zstandard 0.22.0