Lexum: Deterministic Control-Plane Programming for Reproducible Distributed Systems

Aniket Raj

Abstract

Control-plane logic in modern distributed systems suffers from nondeterminism arising from concurrency, asynchronous messaging, and timing variability. This leads to irreproducible failures and difficulty in reasoning about system behavior.

We present **Lexum**, a deterministic control-plane programming model that represents long-lived distributed systems as convergent state machines. Lexum enforces deterministic scheduling, explicit state transitions, logical clocks, and append-only journaling to guarantee reproducibility.

We formalize Ved's execution model, define determinism and convergence properties, and demonstrate its behavior through large-scale stress tests including transaction processing and message storm simulations. Our results indicate that deterministic execution significantly improves reproducibility and system reasoning without compromising scalability.

1 Introduction

Modern infrastructure is governed by control planes such as orchestration systems, workflow engines, and distributed schedulers. These systems continuously reconcile desired and observed state.

However, control-plane logic remains difficult to reason about due to:

- nondeterministic execution
- irreproducible failures
- race conditions and timing dependencies

This violates a fundamental expectation of computation:

Identical input should produce identical execution behavior.

We argue that the root cause lies in the execution model itself. Lexum proposes a deterministic execution model for control-plane systems.

1.1 Contributions

This paper makes the following contributions:

- A deterministic control-plane programming model based on convergence
- A formal execution model with deterministic scheduling
- A runtime architecture enabling replayable execution
- Empirical evaluation using high-load distributed simulations

2 Background and Related Work

Formal Methods: TLA+ models distributed systems as state machines but lacks executable runtime guarantees.

Actor Systems: Erlang provides message-based concurrency but does not enforce deterministic execution.

Event Sourcing: Enables replay but does not guarantee identical execution traces.

Lexum bridges formal reasoning and executable determinism.

3 Programming Model

Lexum programs are composed of domains:

- State S_d
- Goals G_d
- Transitions T_d
- Messages M

Execution proceeds until all goals are satisfied:

$$S_{t+1} = T(S_t)$$

until:

$$\forall g \in G_d,\ g(S) = true$$

4 System Model

A Lexum system is defined as:

$$\langle \Sigma, Q, J, \tau \rangle$$

- Σ: global state
- Q: message queue
- J: execution journal
- τ: logical clock

4.1 Transition Rule

$$m = \min_{\prec}(Q)$$

$$\delta(\Sigma, m) = (\Sigma', M')$$

$$\langle \Sigma, Q, J, \tau \rangle \to \langle \Sigma', (Q \setminus \{m\}) \cup M', J \cdot e, \tau + 1 \rangle$$

5 Determinism

5.1 Definition

Execution is deterministic if:

$$input_1 = input_2 \Rightarrow trace_1 = trace_2$$

5.2 Guarantees

Determinism is achieved through:

- total ordering of messages
- logical clocks
- pure transition functions

6 Convergence

A system converges when:

$$Q = \emptyset \wedge G(S) = true$$

Lexum models systems as convergence processes rather than imperative sequences.

7 Runtime Architecture

Lexum runtime consists of:

- Deterministic scheduler
- Bytecode interpreter
- Mailbox system
- Append-only journal
- Snapshot manager

8 Evaluation

We evaluate Lexum using real execution workloads.

8.1 Experiment 1: Banking Consistency

A domain processes 100 million transactions transferring value between two accounts.

Results:

- All transactions processed successfully
- No state corruption observed
- Execution completed within seconds

8.2 Experiment 2: Message Storm

A producer generates high-volume message traffic to multiple consumers.

Results:

- Stable throughput under sustained load
- No scheduler collapse
- Memory growth remained controlled

8.3 Experiment 3: Multi-Domain Deterministic Synchronization

Three domains exchange messages to reach synchronized target states.

Results:

- Convergence achieved deterministically
- No race conditions observed

8.4 Experiment 4: Convergence Conflict

Conflicting goals with different priorities are introduced.

Results:

- System resolves conflicts via priority ordering
- Converges to stable state

8.5 Experiment 5: Oscillation Case

A system with non-converging transitions is evaluated.

Results:

- System remains stable without crashing
- Highlights need for oscillation detection

9 Limitations

- current implementation is single-node
- external effects introduce nondeterminism
- performance overhead due to journaling

10 Future Work

- distributed deterministic runtime
- formal verification integration
- convergence detection algorithms

11 Conclusion

Lexum proposes a deterministic execution model for control-plane programming. By combining formal reasoning with executable semantics, Lexum enables reproducible, predictable, and analyzable distributed system behavior.