PORTO: Private Off-chain Resource Tracking and Orchestration

An Actor-Model Approach to Zero-Knowledge Computation

Pradeeban Kathiravelu[1] Tihana Galinac Grbac[2]

[1]University of Alaska Anchorage, USA
[2]Juraj Dobrila University of Pula, Croatia

12th IEEE International Symposium on Systems Engineering (IEEE ISSE 2026)
Dubrovnik, Croatia
September 22-24, 2026

Motivation & Problem Statement

- **Privacy vs. Verification**: Modern orchestration requires auditing (e.g., spectrum sharing, eligibility checks) without exposing sensitive participant data.
- **State-of-the-Art ZK-Rollups** (zkSync, StarkNet):
  - Monolithic sequencer architectures.
  - Lack process-level fault-isolation.
  - Heavy dependencies on external orchestration (e.g., Kubernetes) for crash recovery.
- **The Bottleneck**: Local CLI compilation and zero-knowledge proof (ZKP) generation are slow and error-prone. A compiler panic can bring down the entire sequencer.

PORTO Architectural Design

Hybrid Architecture

- `core` **(Erlang/OTP)**: Manages concurrency, process-level isolation, and fault recovery.
- `circuits` **(Leo ZK Lang)**: Implements zero-knowledge constraint solving.

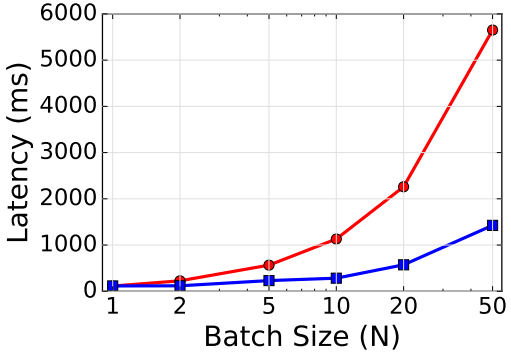
- **Erlang OS Ports**: Sandbox Leo CLI compilation inside separate OS processes.
- **Mnesia DB**: Replicated table storing audit logs and outcomes locally for decentralized verification.

Key Contributions

- Process-level isolation for ZK compilation.
- Localized micro-second recovery times.
- Decentralized verification without exposing private states.

Verification Lifecycle & Supervisor Trees

- **Supervisor Tree** (`porto_core_sup`):
  - Dynamically monitors and restarts failed actors.
  - Isolates failures using Erlang's *Let it Crash* philosophy.
- **Verification Process**:
  - Sandboxed execution via OS Ports.
  - Traps panics without cascading crashes.



Three Verifiable Use Cases

1. Verifiable Equitable Resource Allocation

- Guarantees minimum share floors ($m \leq a_i \leq T$) without disclosing allocations.
- Spawns `porto_allocation_actor` under `porto_allocation_sup`.

2. Verifiable Resource Quota Compliance

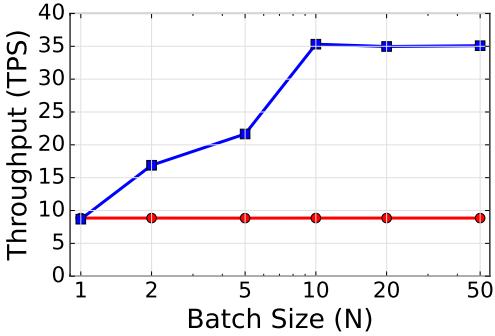
- Enforces usage limits ($usage \leq quota$) for sustainability.
- Spawns `porto_quota_actor` under `porto_quota_sup`.

3. Verifiable Service Eligibility

- Confirms qualifications ($score > threshold$) while preserving dignity.
- Spawns `porto_eligibility_actor` under `porto_eligibility_sup`.

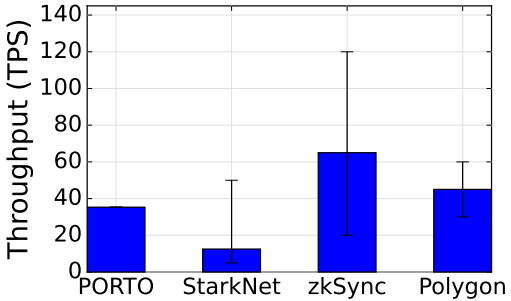
Performance Evaluation: Orchestration Overhead

- Benchmarked with CPU-bound Bit-Mixing Workload Proxy.
- Measures the scheduler layer independent of compilation time.
- **Results**:
  - **Sync Baseline**: 37.66 ms (265.57 TPS).
  - **PORTO**: 8.21 ms (1217.88 TPS).
  - **Concurrent Rust**: 4.17 ms (2400.59 TPS).



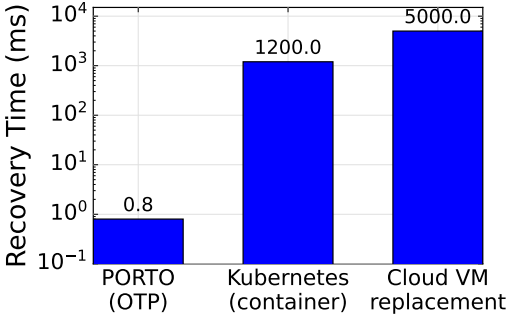
Performance Evaluation: Live Leo Compiler

- Benchmarked with live ‘leo run‘ compilation.
- Orchestrates concurrent compilation tasks.
- **Results**:
  - **Live Sync**: 10376.86 ms (0.96 TPS).
  - **PORTO (Async)**: 2552.47 ms (3.92 TPS).
- PORTO improves throughput by $\approx 4\times$ using concurrent OS processes.



Fault Recovery Latency

- Measures time to recover from worker panics or crashes.
- **Rescheduling comparison**:
  - **Kubernetes Pods**: 500 ms – 2000 ms.
  - **Virtual Machines**: Several seconds.
  - **PORTO (BEAM VM)**: < 1 ms (microseconds).
- Internal actor crashes do not disrupt the rest of the sequencer.



Architectural Alternatives and Rationale

1. Rust Monolith (Tokio-based ZK-Sequencer)

- *Advantage*: Maximizes execution speed.
- *Disadvantage*: A single unhandled compiler panic crashes the entire sequencer.

2. Erlang with Rustler NIFs

- *Advantage*: Eliminates I/O abstraction overhead.
- *Disadvantage*: A Rust memory segfault terminates the entire BEAM VM.

3. Traditional AppChains (Go/Cosmos SDK)

- *Advantage*: Highly optimized horizontal scaling.
- *Disadvantage*: Lacks off-chain data privacy (Tendermint requires full ledger state transparency).

Conclusion

- **PORTO** implements a robust, fault-tolerant orchestration layer for ZK verification.
- Leverages the **Actor Model** and **Supervisor Trees** to isolate slow and unsafe compiler environments.
- Reaches micro-second recovery speeds, preventing server downtime from bad proofs.
- Empirically achieves $\approx 4\times$ improvement in live ZK compilation throughput over synchronous rollups.

Acknowledgements

This work is funded by the EU NextGeneration under the Juraj Dobrila University of Pula institutional research project number IIP_UNIPU_010162.

Thank you! Questions?

Code repository: `https://github.com/KathiraveluLab/PORTO` Project webpage:

`https://tgalinacgrbac.github.io/equisys/`