LP-104: Graph Engine Embedded Subgraph-Compatible GraphQL with SQLite Storage

Zach Kelling

Lux Industries

March 2026

Abstract

We present the Lux Graph Engine, a drop-in replacement for The Graph's Graph Node that eliminates the need for PostgreSQL, IPFS, and WASM subgraph compilation. The engine uses embedded SQLite with WAL mode for storage, native Go resolvers instead of WASM handlers, and serves subgraph-compatible GraphQL queries. Benchmarks on production-scale data (100 tokens, 500 pools, 10,000 swaps) demonstrate 6–200× lower query latency, 2× higher throughput (711 vs. ∼300 queries/second at 100 concurrent clients), and 13× faster event indexing (26,000 vs. ∼2,000 events/second), while reducing memory usage by 40× (50 MB vs. 2 GB) and storage by 25× (2 GB vs. 50 GB per deployment). For the Lux ecosystem's 5 chains across 3 networks, the engine reduces infrastructure from 45 pods to 3, saving approximately $30,000 annually in compute and storage costs alone.

Contents

1 Introduction

The Graph protocol [1] provides a decentralized indexing and query layer for blockchain data. Its reference implementation, Graph Node, requires PostgreSQL for storage, IPFS for subgraph deployment artifacts, and executes subgraph mapping handlers as WASM modules. This architecture introduces significant operational complexity: each chain deployment requires 3 containers (Graph Node, PostgreSQL, IPFS), substantial memory (4 GB minimum), and ongoing database maintenance.

We present a lightweight alternative that achieves subgraph-compatible GraphQL queries with zero external dependencies, embedded in a single Go binary.

2 Architecture

2.1 Design Principles

1. **Embedded storage**: SQLite WAL mode replaces PostgreSQL. No connection pools, no vacuuming, no managed database service.

2. **Native resolvers**: Go functions replace WASM subgraph handlers. No compilation step, no IPFS artifact storage.

3. **Schema compatibility**: GraphQL queries from existing frontends work without modification. Same entity names, same filter syntax, same pagination.

4. **E2E PQ encryption**: All data is streamed to S3 via Replicate with ML-KEM-768 + X25519 hybrid encryption (LP-102).

2.2 Components

Component	Graph Node	Graph Engine
Storage	PostgreSQL (external)	SQLite WAL (embedded)
Artifact store	IPFS (external)	Not needed
Mapping handlers	WASM modules	Native Go resolvers
Schema definition	GraphQL SDL + WASM	Go struct registration
Deployment	3 containers	1 binary
Language	Rust + AssemblyScript	Go

2.3 Resolver Architecture

The engine supports 18 schema types covering all Lux chain types:

Schema	Chain	Entities
AMM	C-Chain	Factory, Token, Pool, Swap, Mint, Burn, Bundle
DEX	D-Chain	Order, Trade, Market, OrderBook
Platform	P-Chain	Validator, Delegator, Subnet, Blockchain
Exchange	X-Chain	Asset, UTXO, AtomicTx
Bridge	B-Chain	BridgeTransfer, LockedAsset, Proof
Threshold	T-Chain	DKGCeremony, KeyShare, ThresholdSig
Privacy	Z-Chain	ShieldedTransfer, Nullifier, Commitment
Quantum	Q-Chain	QuantumStamp, FinalityProof, LatticeSig
Key	K-Chain	Key, Certificate, EncryptionRequest
AI	A-Chain	Attestation, Model, Inference
FHE	T-Chain	FHEOperation, Ciphertext, DecryptionProof
Identity	C-Chain	DID, Credential, Verification
Oracle	C-Chain	PriceFeed, DataRequest, Response
Precompile	C-Chain	PrecompileCall, WarpMessage
Relay	B-Chain	RelayedMessage, AWMReceipt
ServiceNode	P-Chain	ServiceNodeStake, Reward

3 Performance

All benchmarks run on Apple M1 Max with production-scale synthetic data: 100 tokens, 500 pools, 10,000 swaps, 1,000 mints, 1,000 burns.

3.1 Query Latency

Query	Graph Engine	Graph Node	Speedup
Factory (single entity)	10 μs	$\sim$2 ms	200$\times$
50 tokens (list)	398 μs	$\sim$15 ms	38$\times$
50 pools (nested joins)	9 ms	$\sim$50 ms	6$\times$
200 swaps (sorted)	6.5 ms	$\sim$40 ms	6$\times$
Dashboard (combined)	12 ms	$\sim$80 ms	7$\times$

Table 1: Query latency comparison. Graph Node numbers from published p99 latencies on PostgreSQL-backed deployments.

3.2 Throughput

Concurrency	Graph Engine (qps)	Graph Node (qps)
1 client	276	$\sim$100
10 clients	393	$\sim$200
100 clients	597–711	$\sim$300

Table 2: Sustained throughput under concurrent load. Graph Node is limited by PostgreSQL connection pool saturation.

3.3 Indexing Speed

Operation	Graph Engine	Graph Node
Single event write	38 μs (26K/sec)	$\sim$500 μs (2K/sec)
Batch 1000 events	49 ms (20K/sec)	$\sim$2 s (500/sec)
8 concurrent writers	61 μs (16K/sec)	$\sim$1.5 ms (limited)

Table 3: Event indexing throughput. The Graph Engine achieves 13$\times$ higher event processing speed due to zero-overhead SQLite writes vs. PostgreSQL transaction commits.

3.4 Resource Usage

Metric	Graph Engine	Graph Node
Memory (idle)	50 MB	2 GB
Memory (per query)	170 KB	$\sim$2 MB
Disk (per chain)	2 GB	50 GB
Binary size	30 MB	$\sim$500 MB (+ PG + IPFS)
Containers	1	3

4 Cost Analysis

For the Lux ecosystem with 5 chains (C, Zoo, Hanzo, Pars, SPC) across 3 networks (mainnet, testnet, devnet) = 15 deployments.

Component	Graph Node	Graph Engine	Savings
Pods	45	3	42 pods
Memory	60 GB	300 MB	59.7 GB
Disk	750 GB	30 GB	720 GB
Compute ($/mo)	$1,800	$60	$1,740
Storage ($/mo)	$750	$5	$745
Total monthly	$2,550	$65	**$2,485**
Total annual	$30,600	$780	**$29,820**

Table 4: Infrastructure cost comparison for 15 deployments (5 chains × 3 networks).

Combined with the explorer migration (LP-103), the total infrastructure savings are:

Stack	Before ($/mo)	After ($/mo)	Saved ($/mo)
Prior explorer (15 deployments)	$2,400	$0	$2,400
Graph Node (15 deployments)	$2,550	$65	$2,485
Redis (15 instances)	$450	$0	$450
Total	$5,400	$65	**$5,335**
Annual	$64,800	$780	**$64,020**

5 Related Work

LP-102 defines the encrypted SQLite replication standard used for backup. LP-103 defines the omni-chain explorer architecture. The G-Chain VM (`node/vms/graphvm`) provides consensus-backed federated GraphQL for cross-chain queries requiring validator agreement.

References

[1] The Graph, "The Graph: An Indexing Protocol for Querying Networks," https://thegraph.com