Master Project Report

Obsuidian: a fully decentralized RPC solution for the Sui network

Loris Tran and Alexandre Mourot
MSc in Computer Science
École Polytechnique Fédérale de Lausanne

Rachid Guerraoui
Thesis Advisor

Gauthier Voron
Thesis Supervisor

Distributed Computing Laboratory
School of Computer and Communication Sciences
École Polytechnique Fédérale de Lausanne

May 2025

Abstract

This report examines the challenges of Remote Procedure Call (RPC) access in the Sui blockchain ecosystem and proposes a decentralized solution using the Lava Network. We analyze the current limitations of centralized RPC providers, including reliability issues, centralization risks, and lack of economic incentives for Full Node operators. Our solution leverages Lava's decentralized protocol to create a robust, incentivized network of Sui RPC providers, ensuring high availability, censorship resistance, fault-tolerance and economic sustainability. The implementation details, including specification files, provider configuration, and security mechanisms are thoroughly explored and benchmarked to demonstrate the viability of this approach for large scale blockchain RPC infrastructure for the Sui Network.

1 Introduction to Sui Blockchain

Sui is a high-performance Layer 1 blockchain designed for scalable, decentralized applications. Built by Mysten Labs, Sui introduces a new architecture that departs from traditional blockchain designs by implementing a directed acyclic graph (DAG) structure and parallel transaction execution. This architecture enables Sui to achieve high throughput, low latency, and horizontal scalability. Sui's object-centric data model treats on-chain assets as distinct objects with unique identifiers, enabling parallel execution of transactions that operate on different objects.

1.1 Object-Centric Architecture in Sui

Sui implements an object-centric data model that differs from other blockchains. In Sui's architecture, the fundamental unit of state is the object, which is a discrete entity with a unique identifier, owner, and data payload. This object-centric approach enables Sui's parallel execution model, which is central to its scalability proposition.

Objects in Sui are classified into two primary categories with significantly different execution characteristics. First, there are Owned Objects, which are exclusively owned by a single address. These owned objects exhibit several key characteristics that make them particularly efficient to process. Each owned object has exactly one owner (an address) that maintains exclusive control over the object, enabling transactions involving only owned objects from different owners to be processed in parallel without causal ordering constraints. Transactions operating exclusively on owned objects can achieve immediate finality without consensus overhead, requiring only a quorum of validator signatures, which makes these kind of transaction bypass traditional consensus overhead. The execution outcome of transactions on owned objects is also fully deterministic based on the transaction inputs, without dependencies on global state, which will matter latter on for our project.

Then, we also have Shared objects with fundamentally different properties. Multiple addresses can access and modify shared objects, requiring coordination mechanisms to prevent conflicts. Transactions involving shared objects must go through consensus to establish a canonical ordering, as concurrent modifications could lead to inconsistent states. Operations on the same shared object must be processed sequentially to maintain consistency, introducing a synchronization point in Sui's otherwise parallel execution model. The execution outcome of transactions involving shared objects depends on the current state of those objects.

1.2 Full Node vs Validator Node Roles

Sui separates its consensus and its data layer into Full Nodes and Validator Nodes. Validator Nodes are responsible for participating in consensus protocol. These validators execute transactions involving shared objects and collectively maintain the integrity and ordering of such transactions within the system. They act as authoritative sources, issuing certificates for valid transactions, and produce checpoints that represent the resulting state transitions. They are configured to enable quick confirmation of transactions, with a time to finality of arround 300ms.

In contrast, Full Nodes do not participate in consensus but instead focus on maintaining a replica of the global state derived from the checkpoints confirmed by validators. They handle data dissemination and state queries for clients. Full Nodes play a crucial role in enabling scalability and data availability by serving as intermediaries between clients and the consensus layer, caching verified data, and allowing clients to fetch on-chain object states efficiently. Full nodes relay clients transactions requests into Validator Nodes, as can be seen in Figure 1

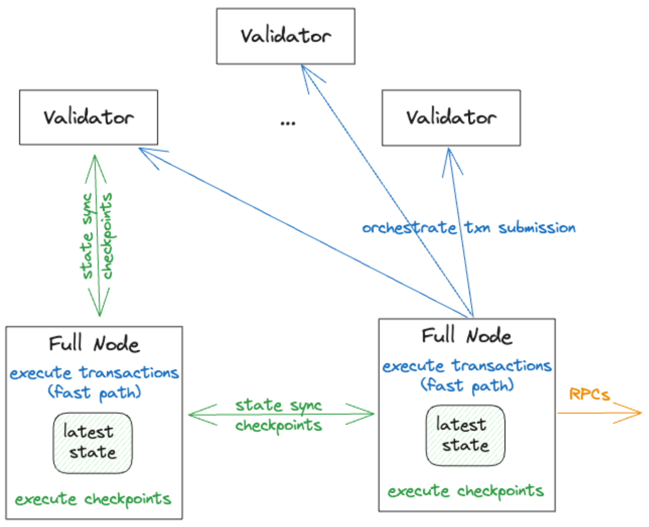

Figure 1: Communication Schema between Full Nodes and Validator Nodes.

The dual object nature of Sui's object model creates specific dependencies on full node state that directly impact scalability. For transactions involving shared objects, full nodes must maintain an accurate and up-to-date view of the object state to support several queries functions. They can return any object stored on the sui blockchain along with its metadata, such as address of the owner, unique ID, etc... Additionally, to provide clients with expected transaction outcomes, full nodes must simulate execution against their current state. For shared objects, full nodes track the state history to ensure correct sequential processing. This state-dependent validation creates a fundamental scalability challenge: as transaction volume increases, the state management burden on full nodes grows correspondingly.

1.3 Full Node Specialization in Data Access

The complexity of Sui's object model necessitates specialized full node configurations to efficiently serve different user queries. Full nodes must maintain additional historical data to support applications requiring access to historical object states. Furthermore, DApp developers heavily rely on transaction simulation capabilities to predict execution outcomes, requiring full nodes to implement simulation logic. These specialized requirements create significant operational overhead for full node operators, while the full nodes have still no incentives to run natively.

In blockchains, scalability remains one of the three cornerstone challenges, alongside decentralization and security, that directly impacts network performance and user experience. To address this challenge, most blockchain networks implement a multi-tiered architecture that separates validator nodes from full nodes, a pattern seen in high-throughput blockchains such as Sui, Aptos, and other next-generation networks. This architectural separation enables validators to focus their computational resources on achieving consensus on the canonical state of the blockchain, while full nodes handle data storage and client accessibility through RPC endpoints. Full nodes can be horizontally scaled to accommodate increasing query loads from applications and users, independent of the consensus layer's scaling constraints, while creating a more efficient network topology where relatively few validators maintain consensus as a larger number of full nodes disseminate blockchain data. Additionally, this separation enhances overall network security by isolating query handling from consensus participation, effectively reducing the attack surface of validator nodes.

However, while this architectural pattern effectively addresses scalability concerns, it introduces a critical dependency: client applications must rely on Remote Procedure Call (RPC) endpoints provided by full nodes to interact with the blockchain. This reliance creates single points of failure, as clients must trust their RPC endpoint for their queries, while full node operators also currently lack direct incentives to maintain their infrastructure.

2 Centralization of the RPC Infrastructure in Sui

The current RPC infrastructure for the Sui blockchain suffers from fundamental structural issues that undermine the network's decentralization principles and create significant operational and security risks. Despite Sui's decentralized architecture, most applications rely on a small number of centralized RPC providers (such as Mysten Labs' official endpoints, Sentio, and Tritton One), creating critical single points of failure that compromise decentralization. This centralization is exacerbated by the absence of economic incentives for Full Node operators, who, unlike validators receiving consensus validation rewards derived from staking rate, have no direct financial motivation to provide reliable RPC services. The lack of economic alignment leads to inconsistent quality of service, with free public RPC endpoints exhibiting variable performance, availability, and data freshness, particularly during periods of high network activity such as token launches and NFT mints.

Centralized RPC providers also introduce censorship vulnerabilities, as seen with incidents where Infura (Metamask) and Alchemy censored access to Tornado Cash contracts, and Venezuelan users of Infura RPC were blocked in 2022 because of legal compliance issues. There are also security concerns, as the current model of trusting individual RPC providers enables potential "man-in-the-middle" attacks where malicious actors could manipulate transaction data, return falsified state information, front-run transactions, or censor specific users while appearing as legitimate infrastructure providers.

Finally, while decentralized RPC solutions exist for Ethereum Virtual Machine (EVM) chains, these are incompatible with Sui's Move-based architecture. Determinism Challenges further complicate matters, as the verification mechanisms used in decentralized RPC networks often rely on deterministic response validation, which becomes more complex in blockchains with different execution models and state representations. Move-based blockchains like Sui introduce Specialized Requirements that require specialized handling in RPC infrastructure.

These issues create significant barriers to Sui's adoption and compromise the network's security and decentralization. Applications built on these platforms must either rely on centralized RPC providers or invest substantial resources in operating their own full nodes, neither of which represents an optimal solution for a decentralized ecosystem. As the ecosystem grows, the demand for reliable RPC access will only increase, potentially increasing centralization pressures and security risks. A decentralized, economically sustainable, scalable, and security-focused RPC solution specifically designed for Sui's architecture is therefore essential for the network.

3 Lava Network: A Decentralized RPC Solution

To address all the RPC challenges in the Sui ecosystem, we propose leveraging the Lava Network—a decentralized protocol specifically designed to provide reliable, censorship-resistant, and incentivized access to blockchain data. Lava Network functions as a decentralized marketplace for RPC services, connecting blockchain applications (consumers) with infrastructure providers through a protocol layer that ensures quality, reliability, and fair compensation. This Network is RPC agnostic and consists of several key components:

Providers are node operators who run Sui Full Nodes and offer RPC services through the Lava protocol, forming the backbone of the network's infrastructure.

Consumers represent applications or services that require access to Sui blockchain data, ranging from wallets and dApps to analytics platforms.

Validators validate transactions on the Lava blockchain and maintain consensus and pairings lists for Providers and Consumers, ensuring the integrity of the network, match them using a pseudo-random algorithm based on stake, geolocation, and quality metrics, optimizing for both performance and decentralization.

Specification Files are governance-approved configurations that define the RPC methods, compute costs, and verification parameters for each supported blockchain, enabling the protocol to adapt to different blockchain architectures.

3.1 Benefits for Sui Ecosystem

Implementing Lava as the RPC layer for Sui offers several significant advantages:

Decentralization is achieved by distributing RPC requests across multiple independent providers, eliminating single points of failure that hinders centralized infrastructure.

Economic Incentives allow providers to earn rewards for delivering reliable RPC services, creating sustainable economics for Sui Full Node operators who previously lacked direct compensation for their services.

Geographic Distribution is enhanced through Lava's geolocation-aware pairing, which ensures consumers can access providers in their region, reducing latency and improving application performance.

Censorship Resistance is strengthened as the random pairing of consumers with multiple providers makes targeted censorship extremely difficult, protecting the network from regulatory or political interference.

Scalability is inherent to the design as demand increases, economic incentives attract more providers to the network, naturally load-balancing capacity to meet growing ecosystem needs, and also support failover to ensure constant uptime when at least one provider is up.

4 Technical Implementation

4.1 Sui Specification File

Sui JSON-RPC Specification

```json
{
  "proposal": {
    "title": "Add Specs: Sui Full Node JSON-RPC",
    "description": "Adding new specification support for relaying Sui Full Node JSON-RPC data on Lava",
    "specs": [
      {
        "index": "SUI_JSONRPC",
        "name": "Sui Full Node JSON-RPC",
        "enabled": true,
        "reliability_threshold": 268435455,
        "data_reliability_enabled": true,
        "block_distance_for_finalized_data": 1,
        "blocks_in_finalization_proof": 1,
        "average_block_time": 1000,
        "allowed_block_lag_for_qos_sync": 10,
        "shares": 1,
        "min_stake_provider": {
          "denom": "ulava",
          "amount": "5000000000"
        },
        "api_collections": [
          {
            "enabled": true,
            "collection_data": {
              "api_interface": "jsonrpc",
              "internal_path": "",
              "type": "POST",
              "add_on": ""
            },
            "apis": [
              {
                "name": "suix_getAllBalances",
                "block_parsing": {
                  "parser_arg": ["latest"],
                  "parser_func": "DEFAULT"
                },
                "compute_units": 100,
                "enabled": true,
                "category": {
                  "deterministic": true,
                  "local": false,
                  "subscription": false,
                  "stateful": 0
                }
              },
              // Additional APIs...
            ]
          }
        ]
      }
    ]
  }
}
```

The foundation of integrating Sui with Lava is the specification file—a detailed configuration that defines how Sui's RPC Methods are exposed through the Lava protocol. Below is an excerpt from the Sui JSON-RPC specification. This specification defines the unique identifier for Sui JSON-RPC services (`SUI_JSONRPC`), security parameters for data verification, minimum stake requirements for providers, supported RPC methods with their compute costs (Compute Units: CU), and determinism classification for each method.

The determinism classification is particularly important for Lava's security model. Methods marked as deterministic should return identical responses across all providers for the same query at the same block height, enabling cross-verification of responses.

4.2 Provider Setup for Sui

To become a Sui RPC provider on Lava, node operators must:

1. **Run a Sui Full Node**: Configure and maintain a Sui Full Node following the official documentation.
2. **Install Lava Software**: Install the Lava protocol software (`lavap`) and configure it to connect to the Sui Full Node.
3. **Stake Tokens**: Stake LAVA tokens to the Sui specification to become eligible for pairing with consumers.
4. **Configure Provider Service**: Set up the provider service to listen for requests and forward them to the Sui Full Node.
5. **Setup SSL**: Set up SSL certificates for the Lava Provider to use.

The provider configuration involves creating a YAML file or using command-line arguments to specify the connection details:

Provider Configuration Example

```yaml
network-address: 0.0.0.0:8080
# Provider Local Listen Address
chain-id: SUI_JSONRPC
# Specification file for Sui
api-interface: jsonrpc
# RPC interface type (here JSON-RPC)
node-urls: http://localhost:9000
# Sui RPC Node URL Address
```

4.3 Docker Compose Deployment

To simplify the deployment and management of the entire Sui-Lava integration stack, we have implemented a global Docker Compose solution. This approach enables operators to deploy the complete infrastructure using a single command, with proper configuration and inter-service communication handled automatically.

The deployment uses a hierarchical Docker Compose structure, with a root `docker-compose.yml` file that orchestrates the entire stack:

Root docker-compose.yml

```yaml
name: master-project

volumes:
  checkpoints_data:

include:

  - path: ./sui_indexer_checkpointTx/
docker-compose.yml
    project_directory:
    ↪ ./sui_indexer_checkpointTx
    name: sui

  - path: ./supabase/docker/
docker-compose.yml
    project_directory:
    ↪ ./supabase/docker
    name: supabase

  - path:
    ↪ ./full-node/docker-compose.yml
    project_directory: ./full-node
    name: full-node

  - path:
    ↪ ./lava/docker/simple-provider/
  docker-compose.yml
    project_directory:
    ↪ ./lava/docker/simple-provider
    name: lava
```

This root compose file leverages Docker Compose's `include` feature to include services defined in the individual project `docker-compose.yml` repositories, while also defining the necessary network connections and dependencies between services.

A `docker-compose.override.yml` file also extends some definitions and acts as a setup on top of the local `docker-compose.yml` file of each repository.

This setup allow composability of the project, as each docker compose project can be run individually, for instance if someone doesn't want to run an indexer, or a database.

Each component maintains its own Docker Compose configuration:

1. **Sui Node Compose**: Configures the Sui Full Node with appropriate volume mounts for persistence and network ports.

2. **Lava Compose**: Sets up the Lava node service along with two provider services and one consumer service, and a nginx proxy. This setup with the stake commands is fully automated and results in having a local Lava Blockchain fork running, which communicates with the two providers and the consumer setup.

3. **Sui Generic Indexer**: Configures the `Sui-Indexer-Alt-Framework` from the Sui Indexer 2.0 platform to ingest its checkpoints from the full node, perform logic, and dump data into a PotsgreSQL database.

4. **Supabase Compose**: Configures a Supabase Docker Instance, an open-source alternative to firebase with a built-in PostgreSQL database, dashboard, analytics, editor, and REST + GraphQL endpoints automatically generated from the database content.

Configuration is managed through a combination of environment files and volume mounts, this envireonnement file is the combination of all the single environnement files from each repository.

.env File Example

```
# Sui Indexer Configuration
POSTGRES_PASSWORD=sui-indexer
START_CHECKPOINT=138216332
LOCAL_MODE=true
CHECKPOINT_DIR=/checkpoints_indexer
PACKAGE_ADDRESS=0x000


# Lava Configuration
LAVAP_VERSION=v5.2.1
LAVAD_VERSION=v5.2.1
PROVIDER_WALLET=servicer1
GEOLOCATION=2
KEYRING_BACKEND=test
LAVA_RPC_NODE=tcp://lava-node:26657
PROVIDER_SUI_PORT=2220

# Supabase Configuration :
POSTGRES_PASSWORD=sui-indexer
JWT_SECRET=xxx

ANON_KEY=xxx
SERVICE_ROLE_KEY=xxx
DASHBOARD_USERNAME=supabase
DASHBOARD_PASSWORD=xxx
SECRET_KEY_BASE=xxx
VAULT_ENC_KEY=xxx
```

The complete deployment process is streamlined to a few commands:

Deployment Commands

```
# Clone repositories
git clone git@github.com:Project-Magma-
    Monorepo/Obsuidian.git

# Configure environment
cp .env.example .env
# Edit .env with appropriate values

# Start the stack
docker compose up -d

# Monitor logs
docker compose logs -f

# Close the stack
docker compose down -v
```

This Docker Compose integration significantly reduces the operational complexity of running a Sui RPC provider on the Lava Network, making it accessible to a wider range of operators and thereby enhancing the decentralization of the RPC infrastructure.

The advantage of this stack is its composability approach. Lava Network allows the creation of a decentralized network of Sui RPC providers. This network offers several benefits over traditional centralized providers. Redundancy is achieved as multiple providers serve the same APIs, eliminating single points of failure that hinders centralized solutions. Geographic distribution is enabled as providers can specify their geolocation, allowing consumers to connect to nearby providers for lower latency, which is important for latency-sensitive applications. The pairing mechanism implements intelligent load balancing by distributing requests across providers based on their stake and capacity, preventing any single provider from becoming overwhelmed during usage spikes. Additionally, automatic failover ensures that if a provider becomes unresponsive, consumers can seamlessly switch to alternative providers in their pairing, maintaining continuous service availability even during partial network outages. Finnaly, regular security checks ensures no single providers can change or corrupt data from a user's queries.

This stack also comes with a custom packet indexer, of which you can see more details in the associted Indexer project report by Alexandre Mourot.

5 Evaluation of Lava Network for Sui RPC Infrastructure

This section evaluates the effectiveness of the Lava Network as a decentralized RPC solution for Sui blockchain. We examine both the economic sustainability through incentive mechanisms and the technical performance through comprehensive benchmarking.

5.1 Economic Incentives and Provider Participation

The Lava Network implements a specififc economic model to incentivize RPC providers and ensure sustainable infrastructure. Anybody can add rewards in any token for RPC providers accross supported blockchains, based on duration and number of request served.

Consumer Subscription Fees form another critical revenue stream, as applications and users purchase subscription plans that grant access to RPC services across multiple chains. These subscription fees flow directly to providers based on their service provision, measured in Compute Units (CUs). Provider rewards are not distributed equally but follow a Quality-Based Distribution system according to several factors. Providers with higher stake receive proportionally more pairing opportunities, incentivizing capital commitment to the network. Those with higher Quality of Service (QoS) metrics—including better uptime, lower latency, and more accurate responses—receive higher rewards, creating a system that rewards providers QoS excellence.

Lava's unique Dual Staking Mechanism allows the same token stake to simultaneously secure the validator network and the provider network, optimizing capital efficiency while maintaining security. This approach reduces the capital requirements for participation while maintaining strong security guarantees. Finally, Slashing Conditions ensure accountability, as providers who deliver incorrect data, experience excessive downtime, or engage in malicious behavior face slashing penalties, creating strong disincentives for poor service. The economic model is designed to create a virtuous cycle: as more consumers join the network, provider rewards increase, attracting more providers and improving service quality, which in turn attracts more consumers.

To evaluate the effectiveness of Lava's incentive model, we analyzed provider participation across multiple blockchain networks supported by Lava.

The data demonstrates robust provider participation across multiple chains, with established networks attracting between 4 to 30 active providers each. This level of participation ensures sufficient decentralization and redundancy for reliable RPC service. The correlation between request volume and incentives shows that the economic model successfully directs rewards to chains with higher demand. The hardware requirements for Sui (16 vCPU, 128 Gb RAM, 4 To NVMe disk) being on the higher end, we estimate the cost to run the Sui Full Node of about 1000$, based on used hardware costs found online. This investment can be financed back by being a provider very fast based on the previous data.

Table 1: Chain Statistics and Provider Rewards

Chain	**# Pr.**	**# Req.**	**Rew.($)**
Near	28	60.72B	26,620
Arbitrum	24	3.09B	15,208
Base	15	3.01B	11,404
Solana	5	5.14B	11,404
Polygon	10	28.94M	11,404
BSC	13	1.01B	11,404
Optimism	14	1.17B	11,404
Ethereum	27	17.6B	11,404
Lava	25	822.68M	9,124
BSC Testnet	5	8.90M	2,273
Base Sepolia	5	4.28M	2,273
Optimism Sepolia	7	4.08M	2,273
Arbitrum Sepolia	12	528.32M	1,513
Cosmos Testnet	4	133.92M	1,133
Movement Testnet	8	24.67M	1,133
Axelar Testnet	30	3.47B	1,133
Stargaze	15	105.27M	373
Evmos	16	4.65B	373

5.2 Benchmarking of the decentralized RPC setup under Lava Network

The benchmarking utilized the `k6_test.js` script using k6, an open-source load testing tool, which implements a structured testing methodology that defines metrics for tracking performance such as Average, Medium, Min and Max Latency, and executes parallel requests to both Lava and centralized RPC endpoints, measures response times and error rates. Tests were conducted using a controlled environment with one local full node running under Docker (Provider 1) and an external RPC provider (Provider 2) using https://sui-mainnet.nodeinfra.com. The test executed 300 RPC calls at a rate of 10 requests per second over a 30-second period.

Our testing framework was designed to compare Lava's RPC performance against existing RPC providers, and measure Lava Network overhead when redirecting existing RPC. We measured **Latency** through response time for various RPC methods: `multiGetObjects`, which fetches multiple objects in a single call, `getLatestCheckpoint`, which retrieves the latest checkpoint information, and `getReferenceGasPrice`, which retrieves the current reference gas price. We also benchmarked the **Throughput** with maximum requests per second without degradation, and the **Consistency** with the variation in response times across multiple requests.

Table 2: Performance Comparison: Lava vs. Centralized Provider

Metric	**Lava**	**Others**	**Diff (%)**
Overall Latency (ms)			
Average	46.20	48.27	-4.28%
Minimum	20.85	24.22	-13.91%
Maximum	122.38	401.86	-69.55%
Median	42.35	35.48	+19.36%
Method-Specific Avg. Latency (ms)			
`multiGetObjects`	49.35	47.60	+3.67%
`getLatestCheckpoint`	42.50	48.49	-12.35%
`getReferenceGasPrice`	46.77	48.73	-4.03%
Error Rate (%)	0.00	0.00	0.00%

Across 300 total RPC calls with a 0% error rate, Lava achieved an average latency of 46.20ms compared to 48.27ms for the centralized provider, representing a negligeable 4.28% performance advantage. While the centralized provider showed a slightly better median latency (35.48ms vs 42.35ms), Lava demonstrated significantly better consistency with a maximum latency of 122.38ms compared to the centralized provider's 401.86ms—a 69.55%.

Method-specific analysis revealed varying performance characteristics across different RPC calls. For the `multiGetObjects` method, Lava was marginally slower with a 3.67% overhead (49.35ms vs 47.60ms). However, Lava demonstrated clear advantages in other methods, with `getLatestCheckpoint` showing a 12.35% performance improvement (42.50ms vs 48.49ms) and `getReferenceGasPrice` performing 4.03% faster (46.77ms vs 48.73ms).

This overall shows there is little to no overhead when running our decentralized RPC setup against a centralized RPC provider.

5.3 Detailed Network Operation Analysis

Examining the consumer docker logs provides details into how Lava's decentralized infrastructure operates during these benchmarks. For the commonly used `getReferenceGasPrice` method, we observed consistent sub-5ms response times from the best-performing providers. Provider 1 demonstrated great performance with response times averaging 2.77ms (ranging from 2.42ms to 3.56ms). This very low latency is explained by this provider running under a local full node, which makes request not have to call an external RPC provider to answer queries. Provider 2 showed higher but still acceptable latencies averaging 19.76ms, with measurements ranging from 18.67ms to 20.85ms. This is explained by this specific provider running under an external RPC provider, which explains the bigger latency since it calls an external URL.

The logs reveal Lava's provider selection mechanism in action during the benchmarking process. The system continuously evaluates providers based on three primary metrics: Latency Score (normalized values ranging from approximately 0.0016 to 0.0089 in our dataset, with lower scores indicating better performance), Availability Score (ranging from 0.80 to 1.0, representing the reliability of provider responses), and Sync Score (a measure of how well providers maintain synchronization with the blockchain's latest state, with values ranging from 0.00067 to over 125.0). These metrics are combined to calculate provider selection probabilities, represented as "shiftedChances" in the logs. For example, at one point the selection probability was distributed as `map[0:0.8703322792862555 1:0.12966772071374455]`, indicating an 87% chance of selecting the 1st provider and a 13% chance for the 2nd, based on their respective QoS metrics.

The consumer logs also demonstrate the network's block synchronization mechanism during our benchmarking. Providers maintained close tracking of the Sui blockchain's latest blocks. The system continuously updates a "finalization information" map that tracks the latest confirmed blocks and their cryptographic hashes, ensuring data consistency across the network. For example, at block 149705486, the hash `CCTDdiV65gUn` was independently confirmed by multiple providers, demonstrating the network's consensus mechanism for validating blockchain data. Block Lag between different providers can vary significantly due to Sui unique Full Node architecture, and this parameter is controlled by the `"allowed_block_lag_for_qos_sync": 100000`, which we have set at an arbitrary value of 100000 for these tests. The Excellence Quality of Service (QoS) metric combines multiple performance indicators to create a comprehensive provider rating. In our tests, we observed that despite having a slightly lower availability score (approximately 0.80 vs 1.0), Provider 1 was frequently selected due to its superior latency performance (1.67ms vs 8.87ms).

Each consumer maintains separate sessions with different providers, identified by unique session IDs (e.g., `sessionId=6586878964949052114`). These sessions persist across multiple requests, allowing for connection reuse and more efficient request handling. Each provider is also assigned a unique identifier (e.g., `providerUniqueId=2322231023700234243`), which is verified on each request by the consumer to prevent provider impersonation. Additionally, block hashes are recorded and compared across providers to ensure data consistency, with entries like: "Added provider to block hash consensus `blockHash=DG58CTFAM1P` blockNum=149705487". This ensures no provider can serve a consumer request if it was not paired with its unique corresponding consumer.

5.4 Fault Tolerance and Provider Failover in Lava Network

To rigorously evaluate the resilience of Lava Network's decentralized RPC infrastructure, we made a test involving the deliberate crash of one provider during our k6 test. This test was designed to simulate real-world scenarios where providers might experience outages due to hardware failures, network issues, or other operational disruptions. The test focused on observing how quickly the system could detect the failure, how effectively it could reroute traffic to healthy providers, and what impact, if any, the failure had on overall service availability.

Before the failure, the provider selection algorithm was distributing requests between both providers based on their performance metrics, and both providers maintained high availability scores, with the soon-to-fail provider showing:

```
availabilityScore=0.931428580000000000
    latencyScore=0.007715430000000000
    providerAddress=Provider1

TRC [Optimizer] returned providers
    shiftedChances="map
    [0:0.9837120524326999
    1:0.016287947567300134]" tier=0
```

Where shifted chances is the probability of going to the first and second provider respectively. When we intentionally crashed one of the two active providers (Here Provider 1 : the local Sui Full Node) during the k6 test, the system immediately began logging connection errors. The first indication of failure appears in the logs with the error messages:

```
DBG could not send relay to provider error
    ="rpc error: code = Canceled desc =
    grpc: the client connection is closing"
     GUID=3558513630266087938 provider=
    Provider1

DBG could not send relay to provider error
    ="rpc error: code = Unavailable desc =
    unexpected HTTP status code received
    from server: 502 (Bad Gateway);
    transport: received unexpected content-
    type "text/html"" GUID
    =8632550552894854902 provider=Provider1
```

These errors indicate that the system was actively attempting to communicate with the failed provider and receiving appropriate error responses, triggering the failover mechanism. After the crash, we observe the system updating the availability metrics for the failed provider. The logs show multiple consecutive relay update entries with success=false flags:

```
TRC [Optimizer] relay update cu=0 latency
    =0s providerAddress=Provider1
    providerData="{Availability:num:
    541.651580, denom: 583.527761, ...
    success=false syncBlock=0}"

TRC GetSessions tempIgnoredProviders="&{
    providers:map[Provider1:{}]
    currentEpoch:2772}"
```

These logs indicate that the system is recording failed relay attempts and adjusting the provider's availability score accordingly. The availability calculation uses a numerator/denominator approach with decay factors, ensuring that recent failures have a stronger impact on the score than older ones. When the score is too low,the second entry shows that the failed provider has been added to the ignorelist for the current epoch (2772). The system then uses this list when selecting providers for subsequent requests, and we see that all requests are being routed to the remaining healthy provider:

```
TRC Choosing providers addon=
    chosenProviders=Provider2 extensions=
    ignoredProvidersList=map[Provider1:{}]
    stateful=0 validAddresses=Provider1,
    Provider2

TRC [Optimizer] returned providers
    providers=Provider2 shiftedChances=map
    [0:1] tier=0
```

This entry confirms that while both providers are still considered valid addresses in the system, the failed provider is being ignored during the provider selection process. The shiftedChances value of map[0:1] indicates a 100% probability of selecting Provider 2, effectively ensuring that all traffic is routed away from the failed node.

When the provider crashed, the system had to establish new sessions with the remaining provider for clients that were previously connected to the failed node. We can observe this process through the creation of new sessions:

```
TRC First time getting providerUniqueId
    for SingleConsumerSession
    providerUniqueId=2322231023700234243
    sessionId=4012650144240411428

DBG jsonrpc http GUID=2546506814101829292
    HasError=false method=POST path="http
    ://sui.obsuidian.xyz/ request="..."
    response="..." timeTaken=4.353097ms"
```

These logs indicate that new sessions are being established with the healthy provider to handle the redirected traffic. The system maintains session state across these transitions, ensuring that client requests continue to be processed without requiring client-side reconnection or retry logic. Despite the failure of one provider, the system maintained its performance metrics. The remaining provider (Provider2) continued to deliver responses with low latency, typically between 2-20ms:

The system's ability to maintain performance during a provider failure demonstrates the effectiveness of Lava's decentralized approach to RPC infrastructure. The temporary nature of the ignorelist (tied to the current epoch) means that if the provider were to come back online in a subsequent epoch, it would be eligible for selection again, subject to its performance metrics. The availability scoring system, with its decay factor **(decay_half_life_time_sec: 3600.000000)**, ensures that past failures gradually have less impact on a provider's score over time. This approach allows recovered providers to gradually regain their position in the network as they demonstrate reliable performance.

Our test setup and benchmarks demonstrates that Lava Network's decentralized RPC infrastructure provides robust fault tolerance through several mechanisms: Quick failure detection through continuous monitoring of provider responses, availability scoring that reflects the provider health status,and an ignorelisting of failed providers to prevent further routing attempts allow efficient traffic redistribution to healthy providers. These mechanisms work together to ensure that even when providers fail, the overall service remains available with minimal performance impact. The decentralized nature of these mechanisms enhances the system's resilience by eliminating single points of failure in the failover process itself. The test results confirm that Lava Network's approach to fault tolerance is appropriate to the challenges of providing reliable RPC infrastructure for blockchain applications, where high availability is critical for supporting decentralized applications and services, while also necessitating no action on the user part.

6 Conclusion

The integration of the Sui blockchain with Lava Network represents a significant advancement in decentralized RPC infrastructure. By addressing the multiples challenges of centralization, reliability, and economic sustainability for RPC providers, this solution enables the Sui ecosystem to scale its queries capabilities while maintaining its decentralization principles.

The technical implementation leverages Lava Network to create a decentralized network of incentivized Sui Full Node operators, ensuring high availability, geographic distribution with load-balancing, redundancy, and fault tolerance. The security mechanisms, including cross-verification and conflict resolution, protect against malicious behavior and ensure data integrity. Our Docker Compose deployment strategy reduces the operational complexity of running a Sui RPC provider on the Lava Network, making it accessible to a wider range of operators with a single line of command to orchestrate the whole stack. This composable approach allows for flexible deployment options, from running the complete stack to selecting specific components based on individual requirements.

Most importantly, the economic model can create sustainable incentives for Sui Full Node operators, solving the fundamental problem of RPC infrastructure funding. As the Sui ecosystem continues to grow, this decentralized RPC layer will become more and more valuable, supporting the next generation of decentralized applications with reliable, fast, redundant, and censorship-resistant blockchain access, and creates a unique economic alignment between RPC consumers and providers.

This approach not only benefits the Sui ecosystem specifically but also demonstrates a viable model for decentralized infrastructure that could be applied to other blockchain networks facing similar challenges.

Acknowledgments

We would like to express our deepest gratitude to Gauthier Voron for his guidance and support as our supervisor throughout this project. We are also deeply thankful to Professor Rachid Guerraoui for accepting our project within the Distributed Computing Laboratory at EPFL, providing us with the opportunity to develop our work in a stimulating academic environment. We would also like to thank the Mysten Lab Team, especially Ashok for his guidance on the Indexer Framework.

Bibliography

References

[1] Mysten Labs, "Sui: A Next-Generation Smart Contract Platform with High Throughput, Low Latency, and an Asset-Oriented Programming Model," Whitepaper, 2022.

[2] Mysten Labs, "Sui Developper Documentation," `https://docs.sui.io/`, 2023.

[3] Mysten Labs, "Sui JSON-RPC API Reference " `https://docs.sui.io/sui-api-ref`, 2023.

[4] Lava Network, "Lava Developper Documentation" `https://docs.lavanet.xyz/`, 2024.

[5] Supabase, "Supabase: An open source Firebase Alternative" `https://supabase.com/docs`, 2024.

[6] Docker Compose, "Docker Compose: Define and Run multi-containers applications on Docker" `https://docs.docker.com/compose/`, 2023.

[7] Grafana k6, "Grafana k6: Load testing for engineering teams" `https://grafana.com/docs/k6/latest/`, 2025.

A APPENDIX: Lava Security and Reliability Mechanisms

A.1 Provider Misbehavior Detection and Types

Lava's security mechanisms address several types of provider misbehavior. Data manipulation involves altering transaction data or responses to benefit the provider or harm users. Censorship occurs when providers selectively refuse to process certain transactions, undermining network neutrality. Eclipse attacks attempt to isolate a consumer by controlling all their provider connections. Denial of service happens when providers refuse to respond or deliberately respond slowly. Inconsistent data provision involves providing different responses to different consumers for the same query, which can lead to consensus failures.

A.2 Cross-Verification Process and Data Reliability

The cross-verification process works through a decentralized verification mechanism to ensure the accuracy and consistency of RPC responses. A consumer sends a request to a provider in their pairing list, and based on the reliability threshold (approximately 1 in 16 requests for Sui), some requests are randomly selected for verification. For these verification requests, the consumer sends identical queries to multiple providers and compares the responses to detect inconsistencies. The specification file includes parameters for block finality and synchronization requirements to ensure data consistency.

Each response from a provider includes a cryptographic signature, creating undeniable proof of the data they provided. This signature serves multiple purposes: it prevents providers from denying they sent a particular response, allows consumers to prove they received conflicting data, creates an audit trail for conflict resolution, and enables the blockchain to verify the authenticity of reported conflicts.

A.3 Quality of Service Monitoring

Lava continuously monitors provider performance through multiple mechanisms. Response time tracking measures the latency of provider responses, while availability checks detect when providers are unresponsive. The system also has consumer reports, allowing users to report quality issues they encounter. These metrics are combined into a Quality of Service Metric (QoS), a score that influences provider pairing probability and ultimately affects provider rewards.

B Conflict Resolution

When conflicting responses are detected, Lava employs a structured resolution process. Consumers can submit a "conflict transaction" containing the original query, multiple signed responses from different providers, block height information, and a consumer signature. This transaction is submitted to the Lava blockchain, triggering the conflict resolution protocol and initiating the dispute process.

The conflict resolution protocol involves a jury process. A set of providers is selected to serve as jurors. The process uses a commit-reveal scheme where jurors first commit to their vote (correct response) using a cryptographic commitment scheme, then reveal their votes along with the commitment opening. Votes are tallied to determine the correct response, and the provider(s) found to have provided incorrect data face penalties.

The jury mechanism is designed to be Byzantine fault-tolerant, meaning it can reach correct conclusions even if some jurors are malicious, as long as the majority are honest. This is achieved through random jury selection to prevent collusion, sufficient jury size to ensure statistical security, economic incentives for honest voting, and penalties for non-participation or provably dishonest voting.

C Penalty Mechanisms

Providers found guilty of providing incorrect data face various penalties:

Slashing: A portion of their staked tokens is confiscated. The severity depends on the nature and frequency of the offense, with repeated offenses resulting in progressively harsher penalties.

Jailing: Providers may be temporarily or permanently removed from the provider pool under certain conditions: providing provably incorrect data, failing to respond to a significant number of requests, accumulating too many negative Quality of Service reports, or claiming more compute units than verified by consumers. Jailed providers cannot receive new pairings until they are "un-jailed" through a governance process or automatic time-based release.

Reputation Impact: Provider misbehavior affects their reputation through reduced Quality of Service metrics, lower probability of being selected in future pairings, decreased attractiveness to delegators, and reduced rewards due to lower compute unit claims. This reputation system creates long-term incentives for honest behavior.

D Economic Incentives

The economic model incentivizes Sui Full Node operators to provide reliable RPC services through multiple reward mechanisms. Compute Unit Rewards allow providers to earn based on the compute units processed for consumers. Subscription Revenue from consumer subscription fees is distributed to providers based on their participation. The dual staking system enables token holders to delegate to both lava validators and individual providers, increasing their effective stake and rewards. Quality Bonuses ensure that providers with higher quality metrics receive more pairings and thus more rewards.

This model addresses the issue of sustainable Full Node operation by creating a viable business model for Full Node Operators. Incentives through public pools helps offset the infrastructure costs of running a Sui Full Node.

E Future Developments

E.1 Sui gRPC Support

As Sui transitions from JSON-RPC to gRPC, we are preparing to support this evolution with several initiatives. A new gRPC specification for Sui is being developed to support the new API format. Streaming support will enhance Lava's protocol to support gRPC streaming for real-time data access. During the transition period, backward compatibility will be maintained with both JSON-RPC and gRPC specifications supported through Lava.

E.2 Kubernetes Deployments

Future enhancements to the setup will also include Kuberntetes files to allow easy deployment and scaling of the docker containers, and will be deployed in several infrastrucure providers such as Amazon AWS, OVH Cloud, or any Kubernetes compatible ecosystem, ensuring geographic distribution and constant availability.