Abstract

In this proposal, I present a method to enhance the efficiency of state management in Ethereum clients, allowing for significant gas limit increase. The proposal involves using a Least Recently Used (LRU) cache to maintain approximately 2.5GB of the most recently accessed state, aiming for a total memory usage of around 8GB. State slots accessed that are part of this cache would priced significantly less gas wise than those outside. This would allow clients like Geth or Reth to make strong assumption for their memory usage and would price disk i/o properly.

This would make 8GB of RAM the minimum requirement to run an ethereum full node.

Introduction

Efficient state management is crucial for Ethereum clients, particularly as the chain continues to grow in size and complexity. A common challenge is maintaining quick access to a large set of state data with limited physical memory. Currently Ethereum’s state is about 100GB, far larger than RAM access of most full nodes. This means we need to price all read/writes high enough in case they were read from disk. The proposed solution introduces an LRU cache at the consensus level, a well-known caching strategy that ensures the most recently accessed data is kept readily available, while less frequently accessed data is relegated to slower storage. With a precise agreement on what should be cached, we can price memory i/o and disk i/o independently, making state growth significantly less of a concern.

Why LRU Cache?

1. Optimizing for Recent Access

: LRU caches are excellent for scenarios where the most recently accessed data is also the most likely to be accessed again soon.

1. Fixed Memory Usage

: An LRU cache can be configured to use a fixed amount of memory (2.5GB of state in this case), making it ideal for systems with limited RAM.

1. Simple and Effective

: LRU is a time-tested caching strategy known for its simplicity and effectiveness in various computing scenarios.

Implementation Details

- State Slot Size

: Each storage state slot is 32 bytes in size.

- Number of Slots

: With 2.5GB dedicated to state data, the cache can hold the latest 83,886,080 state slots accessed by transactions

- New slot access

: When a new slot is accessed, the oldest slot part of the LRU will be evicted and the new slot will be added as the most recent slot accessed in the LRU.

- Recent slot access

: When a slot already part of the LRU is accessed, it will be moved / marked as the most recent state slot accessed.

- Old slot access

: When an old slot is accessed, the oldest slot part of the LRU will be evicted and the accessed old slot will be added as the most recent slot accessed in the LRU.

Advantages

1. Quick Access to Recent State

: Enhances performance for frequently accessed state data.

1. Reduced I/O Overhead

: Limits the need to access slower storage mediums and price them significantly higher

1. Predictable Memory Usage

: Offers a scalable solution that adapts to the changing state access patterns without exceeding the allocated memory.

1. Gas limit increase

: Allows for significantly increasing gas limit due to making state size growth a non-problem, I suspect by 5x to 10x (with 100% of state in memory, a reth client can reach ~20k TPS)

1. Simplicity

: Significantly simpler than other proposals dealing with state growth like ReGenesis, statelessness, state rent, state expiry (this approach is state expiry in disguise), etc.

Potential Challenges

1. Implementation Complexity

: Managing the LRU cache, especially in a multi-threaded environment, requires careful implementation.

1. Memory Overhead

: Ensuring the total memory usage remains within the desired limit needs precise calculation and memory-efficient

1. Minimum RAM requirement

: Clients would now have a minimum RAM requirement of 8GB for full nodes (which seems fine to me in 2024). This can be easily adjusted if we determined at a smaller cache would yield similar benefits (e.g. 128mb).

Example implementation

1. Data Structure

: A combination of a hash map and a doubly-linked list will be used. The hash map provides O(1) access to the state slots, while the doubly-linked list maintains the order of access.

1. Overhead Calculation

: The overhead for the doubly-linked list and hash map is estimated to be around ~5.5GB for these slots, leading to a total memory usage close to 8GB.

From discussion with Remco Bloemen, Georgios Konstantopoulos and Agustin Aguilar