

Plot generation study in Chia Blockchain

By
Shlok Mohan Chaudhari
Vishal Gawade
Prajakta Kumbhar

Introduction

- Chia: a more environmentally friendly cryptocurrency
- Uses proof-of-space-and-time consensus algorithm
- Relies on storage space instead of computational power
- Goal: understand the Plot generation in Chia blockchain

Motivation

- Studying the plot generation process, we can gain insights into the underlying algorithms and data structures used by Chia, which can inform the development of new blockchain systems and protocols.
- Understanding the 4 phases in the Chia plot generation process can help identify potential bottlenecks and areas for optimization, leading to faster plot generation and better system performance.
- Plot generation is a computationally intensive task that requires the optimal use of system resources, including CPU, memory, and storage. Studying the different phases can help identify best practices for resource allocation and management.

Background

- Chia launched in 2017 as a new cryptocurrency and blockchain platform
- Addresses environmental concerns associated with Bitcoin
- Uses a more energy-efficient consensus algorithm
- Plot generation speed remains a challenge
- Requires significant disk operations for plot generation



Proposed solution

Forward Propagation:

Compute all tables and final outputs f_7 .
Temporary file allows proof creation but is not space-efficient

Backpropagation:

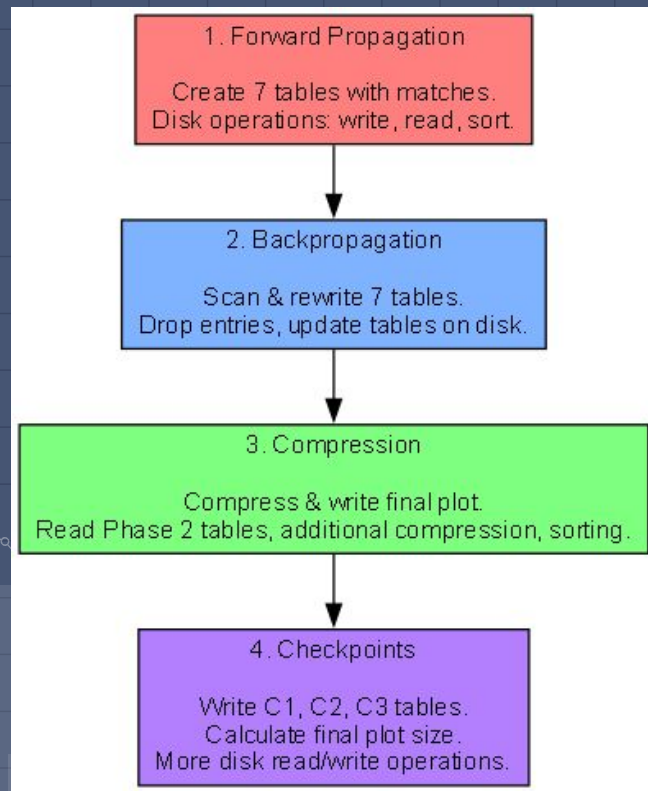
Remove unuseful data for finding proofs.
Adjust positions in next table for dropped entries.

Compression:

Convert from (pos, offset) to double-pointer format.
Reduce entry overhead for more compressed storage.

Checkpoints:

Iterate through Table_7 and write Table_7.
Compress tables into checkpoint tables.



Algorithm

Forward propagation phase:

```
For x in  $0 \dots 2^k - 1$ :  
    Compute  $f_1(x)$   
    Write  $(f_1(x), x)$  to table1  
For table in 1..6:  
    Sort tablei by  $(f_i(x), \text{pos}, \text{offset})$ . for table1, by  $f_1(x)$   
    For entry in tablei:  
        if entries L and R match:  
            Compute  $f_{i+1}(\text{CL}, \text{CR})$  for table1,  $M=x$   
             $C = \text{Collation}_i(\text{CL}, \text{CR})$   
            Store  $(f_{i+1}, \text{pos}, \text{offset}, C)$  in tablei+1
```

2 Backpropagation phase

```
For table in 6..1:  
    Iterate through tablei and tablei+1:  
        Drop unused entries in tablei  
         $\text{sort\_key} = \text{table} == 7 ? f_i : \text{table}_{i+1}\text{pos}$   
        Rewrite used entries in tablei as  
         $(\text{sort\_key}, \text{pos}, \text{offset})$   
        Rewrite entries in tablei+1 as  $(\text{sort\_key},$   
         $\text{pos}, \text{offset})$   
    If  $i > 1$ :  
        Sort tablei by  $(\text{pos}, \text{offset})$ 
```

Algorithm

Compression phase:

For table in 1..6:

 Iterate through table_i and table_{i+1}:

 Read sort_key, pos, offset from table_{i+1}

 Read table_i entries in that pos and offset: eL, eR

$y, x = \text{sort}(eL.\text{newPos}, eR.\text{newPos})$

$\text{line_point} = x * (x - 1) // 2 + y$

 Rewrite table_{i+1} entry as (line_point, sort_key)

 Sort table_{i+1} by line_point

 For entry e, i in enumerate(table_{i+1}):

 newPos = i

 Rewrite e as (sort_key, newPos)

 Write compressed e.line_point deltas to table P_i

 Sort table_{i+1} by sort_key

Checkpoints phase:

For entries in table7:

 Compress f7 entries into C1,C2,C3 tables

 Write pos6 entries to table P7k

Evaluation

Phase 1: Create seven tables with different matches

Disk operations: writing, reading, and sorting data

Phase 2: Scan and rewrite each table from Phase 1

Disk operations: read original tables, drop entries, write updated tables

Phase 3: Compress and write final plot to disk

Disk operations: read Phase 2 tables, compress and sort, write final plot data

Phase 4: Write additional tables (C1, C2, C3) and calculate final plot size

Disk operations: read and write operations for additional tables

Copying the final plot: Copy plot to destination directory

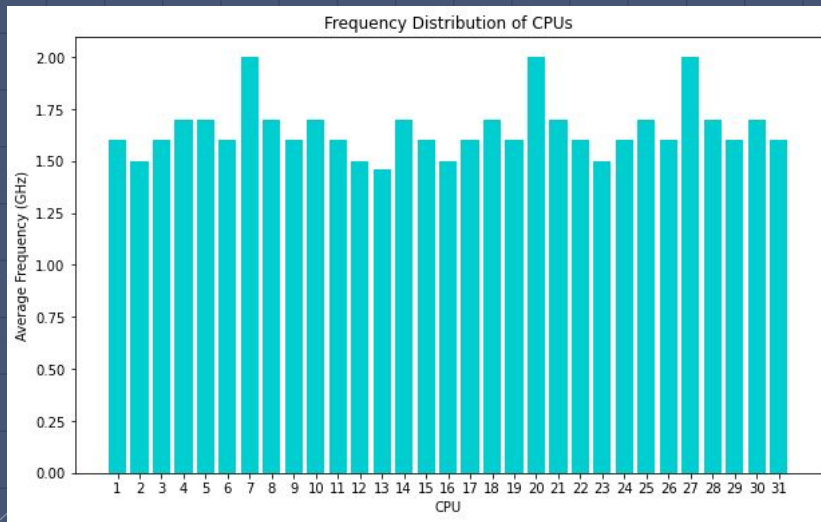
Disk operations: read plot from temp directory, write to destination directory



Evaluation

Frequency Distribution:

- CPU frequencies range from 1200 MHz to 3.00 GHz.
- Idle percentages vary across CPUs, with some having higher idle percentages.
- The idle percentage of CPUs varies from 0.3% to 2.7%
- Most CPUs have the majority of their frequency distribution in the 3.00 GHz range.
- Exploration of more efficient frequency scaling policies can lead to better power management.



Related work

Bram Cohen's paper "Chia: A better bitcoin"

- Proposes a new cryptocurrency
- Utilizes proof-of-space-and-time consensus mechanism

The chia-plotting-optimizer GitHub repository (<https://github.com/codez0mb1e/chia-plotting-optimizer>)

- Provides an open-source tool to optimize Chia plot generation performance

Dinh et al's paper "Untangling Blockchain: A Data Processing View of Blockchain Systems"

- Provides a comprehensive overview of blockchain systems from a data processing perspective

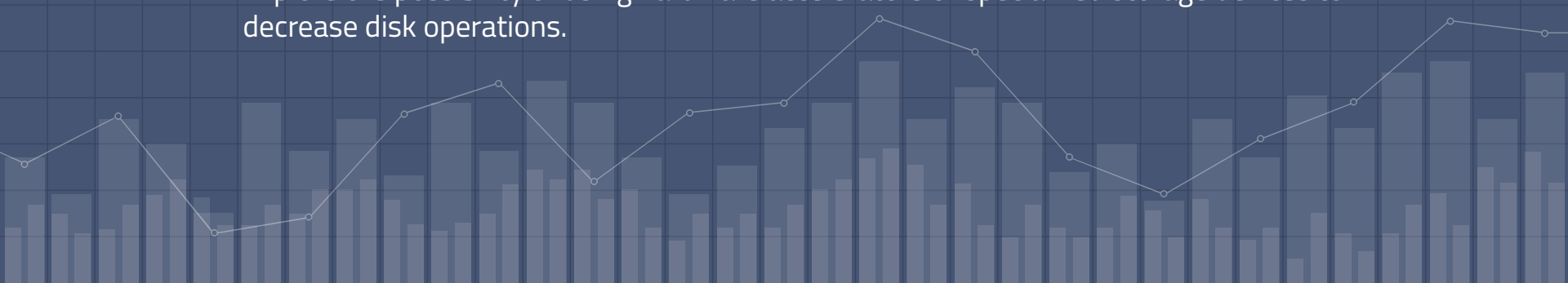
The Chia website's "Plotting Basics" page

- Offers an introduction to the Chia plot generation process.

Conclusion

- Since the plot generation requires iterative searching and sorting algorithms, the plot generation performs such repetitive tasks, which results in overhead.
- the system is experiencing a high number of disk write operations, specifically in the writeback process, which are power-intensive and may be slowing down the plot generation process.

Future work:

- Further optimize CPU utilization by enabling better load balancing
 - Investigate the feasibility of implementing parallel processing techniques.
 - Explore the possibility of using hardware accelerators or specialized storage devices to decrease disk operations.
- 

Thank You!

