様式６

申請先学部　工　学部　採択番号　No.03

令和４年度大阪大学未来基金「学部学生による自主研究奨励事業」研究成果報告書

ふりがな 氏名	チョウ　タクホウ ZHANG　ZEPENG （張　澤澎）	学部 学科	工学部 電子情報工学科	学年	３年
ふりがな 共同 研究者氏名		学部 学科		学年	年
					年
					年
アドバイザー教員 氏名	鬼塚　真	所属	情報科学研究科		
研究課題名	高速なブロックチェーンエンジンの研究				
研究成果の概要	研究目的、研究計画、研究方法、研究経過、研究成果等について記述すること。必要に応じて用紙を追加してもよい。（先行する研究を引用する場合は、「阪大生のためのアカデミックライティング入門」に従い、盗作剽窃にならないように引用部分を明示し文末に参考文献リストをつけること。）				

1. 研究の目的

近年、インターネット技術の発展と伴い、個人情報プライバシーの保護及びデータ所有権の分散化が話題になりつつある。それに携わり、仮想通貨、分散型アプリケーション（Dapps, Decentralized APPlicationS）及び分散型自律組織（DAO, Decentralized Autonomous　Organization）など、データを暗号化、またはその所有権を分散化して、個人情報プライバシーをより一層守る技術が相続いで流行ったり、インターネットの発展に活力を注いでいる。

しかし、これらの技術は、どんな美辞麗句を連ねても、ブロックチェーンエンジンに基づき、開発されるブロックチェーンシステムの実用だけである。すなわち、近年に流行ったインターネット技術に関わるそれぞれの概念は、ブロックチェーンシステムを基盤として実例化されるものだけである。それゆえ、私はそれぞれの新たなネット技術を究明したいため、高速なブロックチェーンエンジンの研究をする。

本研究では、ブロックチェーンシステムの根本を究明し、その基幹となるブロックチェーンエンジンを Rust 言語により実装し、ベンチマークにより性能評価されてなるべく高速化のように改装する。

2. 研究計画・研究方法

ブロックチェーンシステムのアーキテクチャーの設計

ブロックチェーンシステムの根幹技術は、実際に分散データベースシステム及び P2P ネットワークと同じである。それゆえ、先に分散データベースシステム及び P2P ネットワークの原理を究明し、さらにその設計法を学び、ブロックチェーンシステムのアーキテクチャーを設計する。

Rust 言語により実装

Rust 言語とは、C＋＋と同じように効率性よく、かつ C＋＋よりエコシステム及び安全性が極めて優秀なシステムプログラミング言語である。以前の分散システム及びブロックチェーンエンジンはほとんど C＋＋により実装されたが、最近には Rust により実装されるのも少なくないである。私は、なるべくバグなしのようにブロックチェーンエンジンを実装したく、Rust 言語も深く学びたいため、Rust 言語によりブロックチェーンエンジンを実装する。

性能評価

Warp というターミナルアプリの実行時間測定機能を用いて、一万個トランザクションのベンチマークにより時間を測りながら評価される。

3. 研究経過

アーキテクチャーの設計

階層ごとにブロックチェーンエンジンのアーキテクチャーを設計する。ブロックチェーンエンジンは、分散データベースの特化であるため、アーキテクチャーが分散データベースとよく似合う。それゆえ、本研究は、既存の分散データベースシステムの設計を参考し、ブロックチェーンエンジンのアーキテクチャーを設計する。コアとなるのは、ストレージ・エンジン層及びコンセンサス・エンジン層である。本研究も、ストレージ・エンジン及びコンセンサス・エンジンの実装及び高速化に専念してアーキテクチャーを設計する。

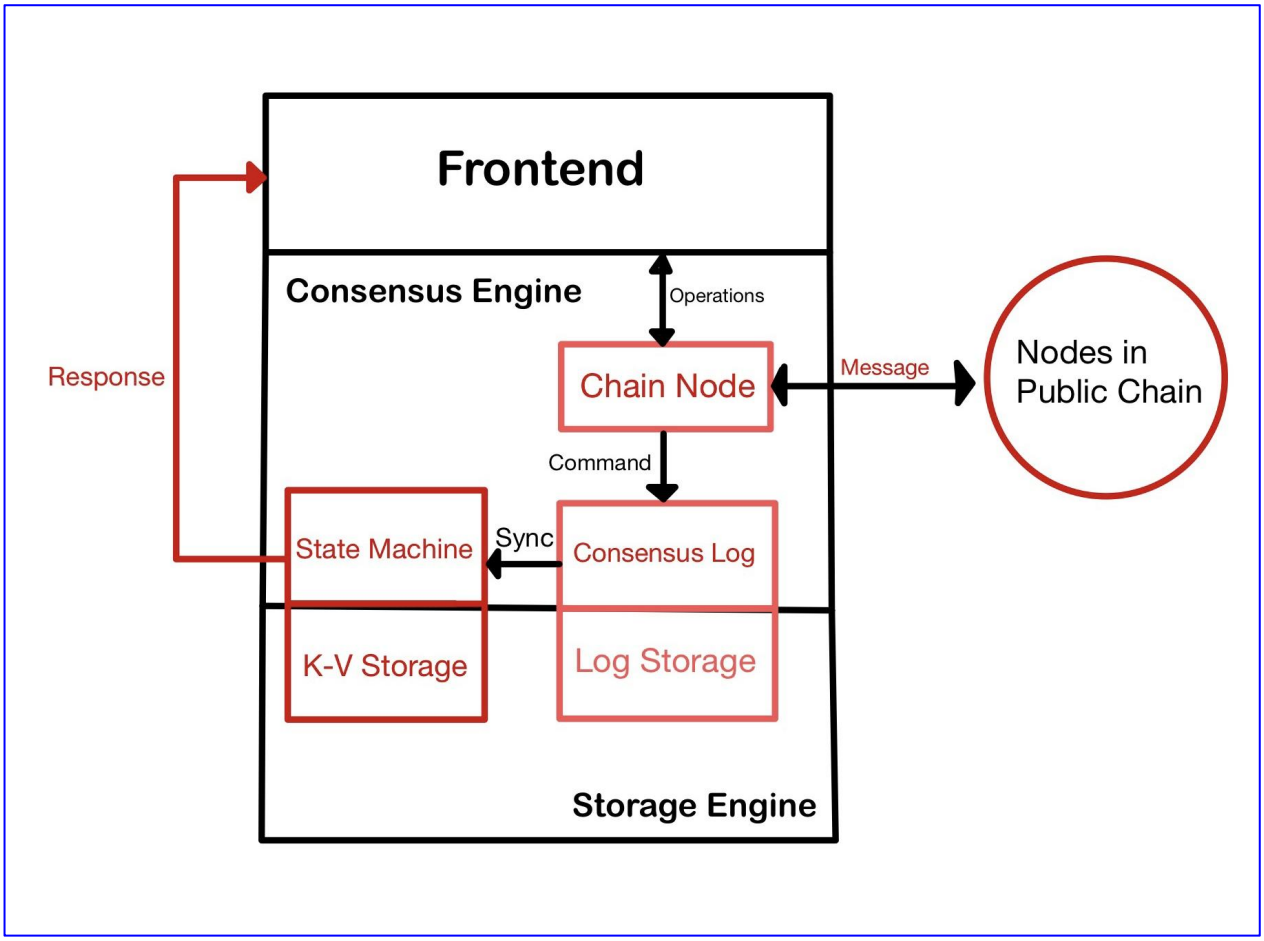

Figure 1. アーキテクチャー設計図

ブロックチェーンエンジンの実装

アーキテクチャー設計図に従い、各レイヤの実装についてデータ構造を設計し、実装法を考える。

私は、効率性及び安全性を鑑み、ストレージ・エンジンを LSM　Tree で、コンセンサス・エンジンを Raft で、実装される。以上のコア部分は基本的に論文を読みながら、Rust 言語によりそのデータ構造及びメソッドを再現して実装される。

最後、ストレージ・エンジン層及びコンセンサス・エンジン層に公有したメソッドに基づいて、インタフェースを設計し、CLI（CommandLine　Application）の形式で実装する。

Log–Structured Merge–Tree[1]の実装及び高速化

LSM Tree の論文誌を読みながら、ブロックチェーンシステムに大切な読み書き性能、他のノードと同期化及びその際のデータ操作安全性を考え、ストレージ・エンジン層モジュールの根幹となる LSM　Tree のデータ構造及び API を設計し、BloomFilter により読み書き性能の最適化をしながら実装する。

```
pub struct LSMTree {
    levels: Vec<Level>,
    buffer: Buffer,
    worker_pool: threadpool::ThreadPool,
    // used for Bloom filter
    bloomflter_bits: f32,
    depth: u64,
}
```

Figure 2.　主なデータ構造

```rust
pub fn new(
    buf_max_entries: u64,
    depth: u64,
    fanout: u64,
    bloomfilter_bits: f32,
    num_threads: u64) -> LSMTree;

pub fn get(&self, key: &str) -> Option<String>;

pub fn put(&mut self, key: &str, value: &str) -> bool;

pub fn delete(&mut self, key: &str);

pub fn close(&mut self);

pub fn open(&mut self, filename: &str);

pub fn range(&self, start: &str, end: &str) -> Vec<String>;
```

Figure 3.　LSM Tree の API 設計

初めて実装の際には、読み書き性能は予想ほどよくないと考えるので、高速化に工夫する。性能を抑える問題点は主に以下である：

・ディスク・ストレージをメモリにブロックキャッシュ（BlockCache）する際には、その get_block_cache()関数の実行速度が遅くなる。

この問題点は、get_block_cache()関数の中に全部の for 文を Rust の関数型プログラミング特性 Iterator を用いて改めて書き直したら、実行速度が早くなり、高速化される。

Raft[2]プロトコルの実装

Raft の論文誌を読みながら、主に State　Machine 及び Raft　Log というデータ構造及びそれらのメソッドの設計に工夫しながら Rust 言語により実装する。残念ながら、

私は実装した Raft モジュールは、単独でよく動けるが、ストレージ・エンジン層と組み合わせながら実行すると、正確に動けなくなった。私は最後までもこの問題をデバッグして動かせるように努力したが、やはり解決されなくなった。それを鑑み、私は、コンセンサス・エンジンの部分には、オープンソース・コミュニティのパッケージを使いながら、ストレージ・エンジン層と組み合わせて実装をする。

CLI インタフェースの実装

Rust コミュニティの Clap[3]パッケージを使いながら、ストレージ・エンジン層及びコンセンサス・エンジン層と組み合わせて実装する。

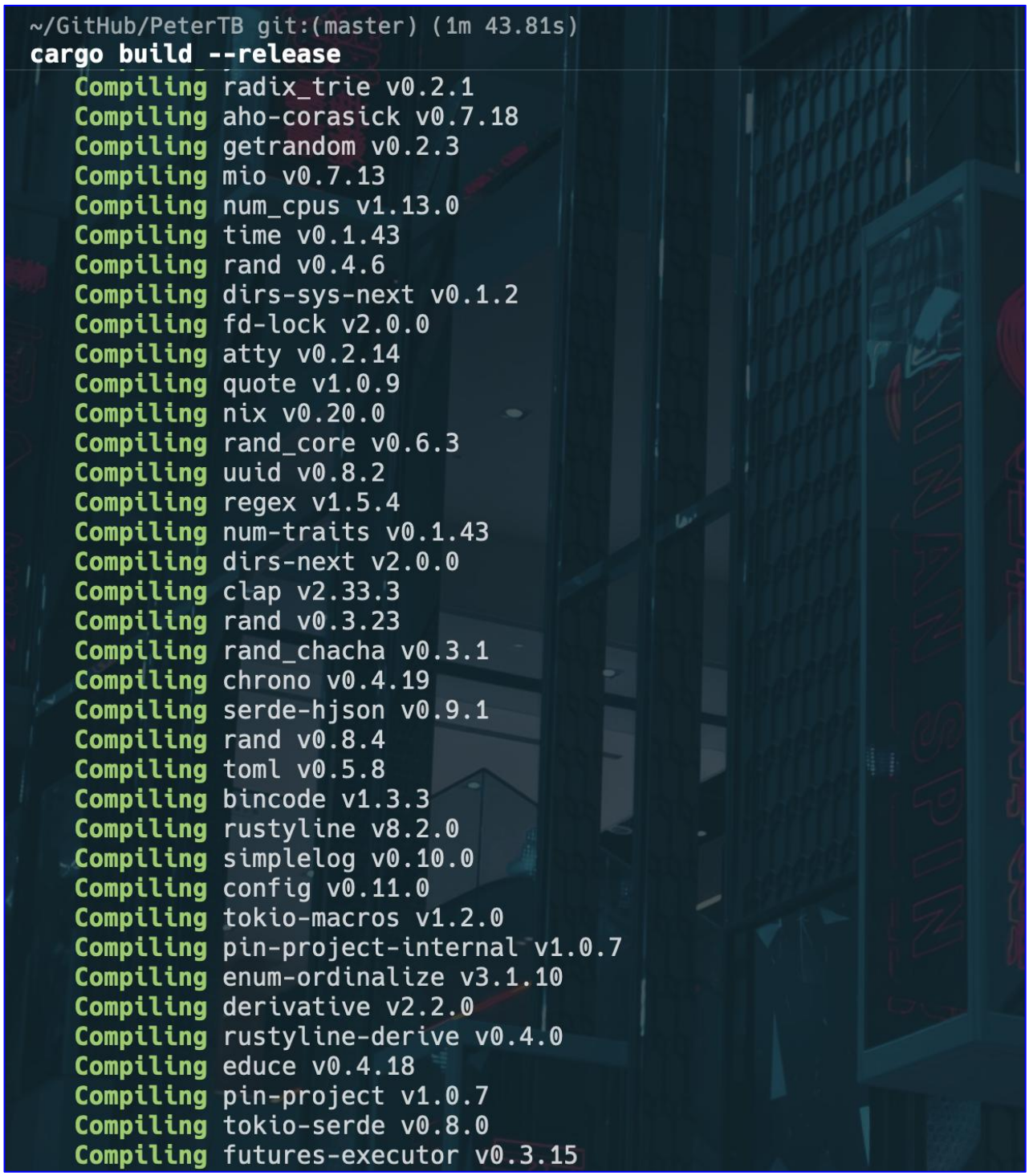

Figure 4. コンパイル中の様子

4. 研究成果

ブロックチェーンエンジンは、実質分散データベースシステムの特例化であることがわかった。また、分散データベースシステムの設計法に基づいて、ブロックチェーンエンジンをアーキテクチャーの設計から Rust 言語の実装までの開発が成功した。4 ヶ月の間におよそ２万行の Rust コードを書き、テスト及びデバッグしたのも極めて貴重な経験だと思われる。

残念ながら、自作の Raft モジュールがいまだに幾つかの不具合があり、まだブロックチェーンエンジンに組み合わせない。また、高速化の工夫も実装に結構時間かかったので、ほとんどやらなかった。報告書はここまでしか書かなかったが、今後にも Raft モジュールの不具合を解決し、LSM Tree に基づいたストレージエンジンの高速化に工夫すると思われる。

本研究のコード、レポート及びドキュメントは、以下のレポジトリに置く。今は結構プロトタイプのようなプログラムしか出来上がらないので、一旦不公開となる。今後には Raft モジュールのバッグを解決していければ、ドキュメントをより詳しく書き上がって公開しようと思われる。

本研究に関わるレポジトリ

github.com/PeterWrighten/PeterResearch

参考文献

[1] *O'NEIL, P., CHENG, E., GAWLICK, D., AND O'NEIL, E.,* The Log-Structured Merge-Tree (LSM-Tree). *Acta Informatica 33, 4(1996), 351-385*

[2] *Ongaro, Diego; Ousterhout, John (2013).* In Search of an Understandable Consensus Algorithm(Extended Version)

[3] Clap Package, *doc.rs/clap*