PepperDB: Self-Evolving Database for Pure DApps

Roy Guo, Sean Fu, Remy Trichard, Peng Lei

www.pepperdb.org

ABSTRACT

PepperDB is a decentralized database that makes DApp developers build DApps much easier and lower down blockchain storage cost tremendously, and with our Pepper Rank (Inspired by Google PageRank) system, PepperDB is able to self-evolving continuously.

PepperDB provides a SDK that supports SQL/NoSQL operations directly on blockchain, which release developers from coding and testing complex smart contracts.

PepperDB also achieved best data compression ratio than anyone else in the market, which has already been proven inside Alibaba Cloud (Alibaba bought the license of TerarkDB storage engine, which is an implementation of our technology). Our Searchable Compression technology can access data directly on highly compressed data file, which means it is much faster than ever (No decompression cost).

PepperDB also supports fully distributed applications via a DApp Store that is embedded into wallet. Nowadays, almost all existing DApps were deployed on centralized servers to handle user requests, and help users to store / query blockchain data. This is because most blockchain wallets doesn't support client side running environment, in which case we believe is not a safe enough (users still need to trust these centralized servers).

This project is trying to put all of these together and make blockchain a better platform for the world.

1 INTRODUCTION

Lots of blockchain-based applications are emerging, but we have identified that some critical problems still have not been solved properly.

DApp Development is Painful. One of the most famous blockchain that supports smart contract development is Ethereum. If a Ethereum developer wants to do something special rather than to make a simple token or a simple crowdsale, he is likely to create his own smart contract. But a serious Ethereum smart contract development requires a programming language like Solidity or Javascript, a Ethereum client like Ganache, a test / debug / deploy environment like Truffle, and some new developers may need a secure smart contract library like OpenZeppelin. This is really painful, so PepperDB wants to make it easier and provides a set of SDKs that allows developers to store or query data from blockchain directly using SQL / NoSQL schema. And for most of cases, smart contract programming is no longer needed (But is still supported).

Block Data Grow Too Fast. We all know that blockchain data is growing very fast and there seems no efficient way to reduce the growth speed. Some people turn to new path like sharding (Ethereum), off-chain storage (IPFS) or centralized server groups (EOS). Each of these methods has their advantages and disadvantages, we choose to use off-chain storage in the future (after it has been proven to be secure). But for the data itself, we are going to make them grow slower than ever. Our cutting edge Searchable Compression technology could compress the whole blockchain dataset into one single block, which can significantly reduce the total disk size of the data. From tests in different scenarios, we achieved 3 to 10 time better compression than other compression algorithms. And with this great compression, we can extract data directly on compressed data file, which has never been done before.

Most DApps are Still Using Centralized Server. Most of the world's DApps are using deployed in a centralized server now. These centralized servers are used for handling user requests and interact with blockchain. This is not pure DApp because users still need to trust these centralized servers and these server could been hacked or service down. And this also requires developer to handle server security, user privacy properly, which is not a simple task for

most people. PepperDB is trying to build a completely distributed application platform via a DApp store and native client side support. Users can download DApp and run it directly on their own machine. Since we have best compression and these DApps don't need to cached in every node, so the cost of storage is acceptable under our technology.

Blockchain is Relatively Hard to Upgrade. We believe all softwares need updates, including blockchain. Small or simple updates in blockchain are easy, but for those critical updates like migrate from PoW to PoS, could hurt some participants, which sometimes could leads to a hard fork. But for a blockchain database, we understand that a lot of work hasn't be done properly (e.g. the off-chain storage mechanism is still not proven to be secure), so a strategy that allows us to upgrade important components is very useful. To make this happen and avoid hard forks as much as possible, we are designing a new voting system based on user contribution. This means when a user is more valuable (instead of hold more tokens) to the network, he will has more power to influence the result. User contribution is calculated based on their connection with each other (We call it Pepper Rank, inspired by Google PageRank).

2 RELATED WORK

High Level Architecture. We want to make things as simple as we can, and the actual implementation will be done step by step. Here is our highest level design:

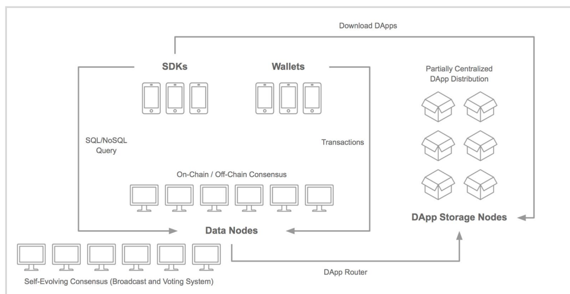

Figure 1. PepperDB High Level Architecture

It includes several important parts: ***SDK*** is a library that developers use inside their DApps, it helps to handling data writing and reading. It's a replacement of most smart contract. ***Wallet*** is used for transactions or voting. And it also takes the responsibility of DApp store, which helps users download DApp and run it directly on their local machine. ***Data Node*** is traditional blockchain working node. We will split all nodes into on-chain and off-chain nodes in the future. ***DApp Storage Node*** is used for storing DApp itself and DApp version history. These data don't need to put into blockchain and requires fast access or download.

2.1 Database Protocol

SQL is the most widely used database query language in the world today, and NoSQL is catching up these years. They are used for different scenarios so we would implement both of them in the future. But the first step would be a simpler one, NoSQL.

Leverage database protocol into blockchain is actually very easy. A traditional database generally has a architecture like this:

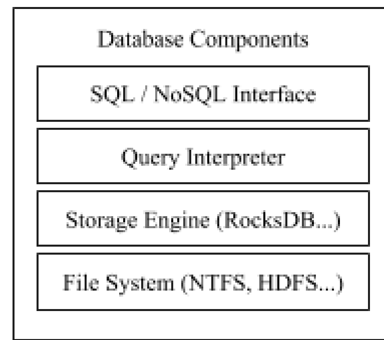

Figure 2. An Ordinary Database Architecture

If we change the lowest part, file system into blockchain, a basic version of blockchain database is done. But as a fully distributed system, we need to handle data secure, query efficiency and other critical issues. Relying our understanding of database technology, we re-designed the blockchain database within this architecture:

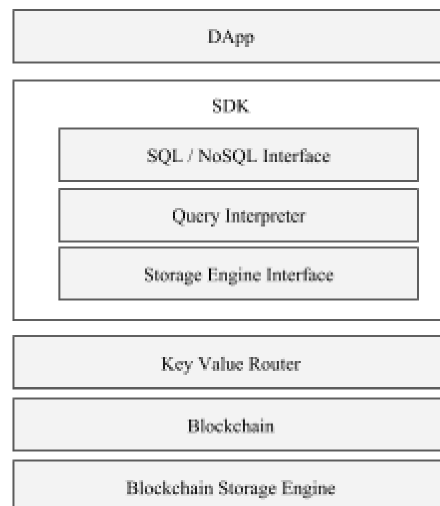

Figure 3. Blockchain Database Architecture

2.2 Searchable Compression

The Searchable Compression technology is originally invented by ourselves and has been successfully used inside Alibaba Cloud. The main purpose of this technology, is to compress as much data as we can into one single block, make the compression ratio approximately to the its limit and improve access speed in the same time. Most of people understand that if someone wants a better compression, it has to compress more data together but lose access speed since it requires decompress before accessing, in other words, its a trade-off. But for our searchable compression, its not, we improved both compression ratio and access speed tremendously (compression ratio improved by 3 ~ 10 times, access speed improved by 10 ~ 200 times).This technology is composed by two algorithms, CO-Index and PA-Zip. As we all know, almost all storage systems (e.g. database, blockchain itself, file system) has a key-value storage engine layer. In our case, CO-Index is for key / index compress, PA-Zip is for data / value compress.

2.2.1 CO-Index (Compressed Ordered Index). Ethereum is using LevelDB as storage engine, Nebulas is using RocksDB, and traditional database MySQL is using InnoDB. These famous storage engine uses different index compression algorithms but their common point is, the compression ratio is not good. CO-Index uses three different methods to improve its compression ratio: ***Succinct Data Structure*** is a memory efficient data structure. Comparing pointer-based structure, it only uses 1/32 to 1/64 memory to represent a tree structure. ***Patricia Trie*** is a path compression Trie. It compresses all single-child nodes into one node, save a lot of space when meeting complex string data. ***Nested Succinct Patricia Trie*** improves the compression ratio even better. It uses all the compressed path that patricia trie generated and compress them into a new Trie, then nest it into previous one.

Here's a the basic idea of how Patricia Trie works (Size and Path Length of Patricia Tries: Dynamical Sources Context):

With any finite set X of infinite words produced by the same source, we associate a trie, $Tr(X)$, defined by the following recursive rules:

(R_0) if $X = \emptyset$, Then $Tr(X)$ is the empty tree.

(R_1) if $X = \{x\}$ has a cardinality equal to 1, then $Tr(X)$ consists of a single leaf node represented by x,

(R_2) if X has a cardinality of at least 2, then $Tr(X)$ is an internal node represented generically by • to which r subtrees are attached,

$$Tr(X) = <\bullet, Tr(T_{a_1}X), Tr(T_{a_2}X), ..., Tr(T_{a_r}X)>$$

The edge attaching the subtrie $Tr(T_{a_j}X)$ is labeled by the symbol a_j.

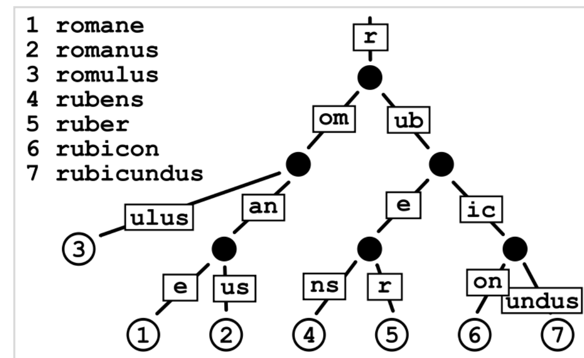

Figure 4. Patricia Trie and Path Compression

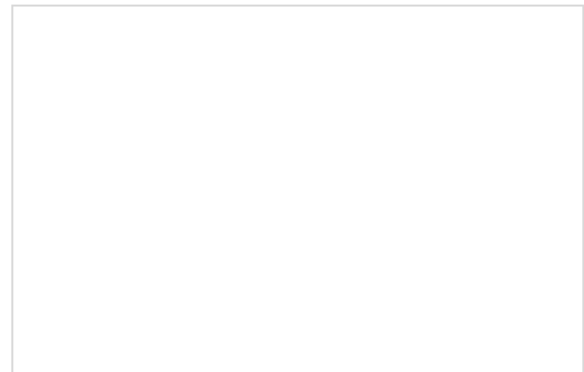

Figure 5. Generate new Patricia Trie by Compressed Paths

2.2.2 PA-Zip (Point Accessible Zip). All existing database compression algorithms are using block-based compression. Its basic idea is to compress data block by block (e.g. 16KB block). The reason that people are doing this is because they have to decompress it before reading the data. So a

bigger block means better compression but lower access speed. This kind of trade-off optimization is all over the internet. PepperDB solved this problem by providing a searchable compression technology. PA-Zip is not block-based, it can compress all data into one single file (or you can call it a giant block), and no need to decompress it before reading.

2.2.3 Benchmarks. Our algorithms had been tested by some of the biggest internet companies in the world, including Alibaba, Baidu etc. Here are some results:

1) Bare metal storage engine test (CPU 16 cores, 64 GB memory, TPC-H 550GB dataset):

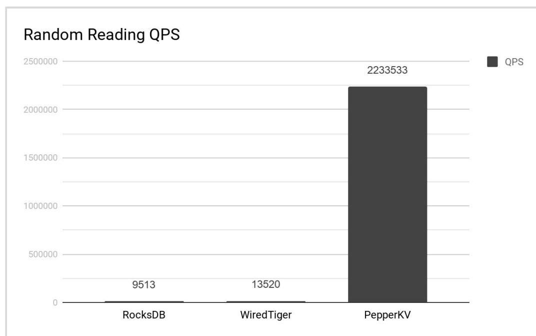

Figure 6. Random Reading on TPC-H

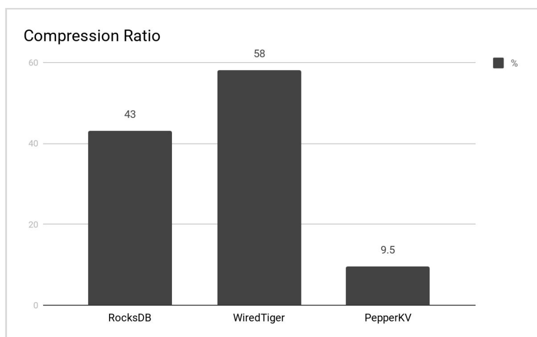

Figure 7. Compression Ratio on TPC-H

2) Test results from Alibaba

• About 3.8 TB of raw data

• After compression with TerarkDB's algorithm, data is compressed to about 1.1 TB

• With outstanding compression ratios, the reading performance is also 3 to 5 times better

• This scenario is not our best scenario, and other engines are highly optimized via Alibaba

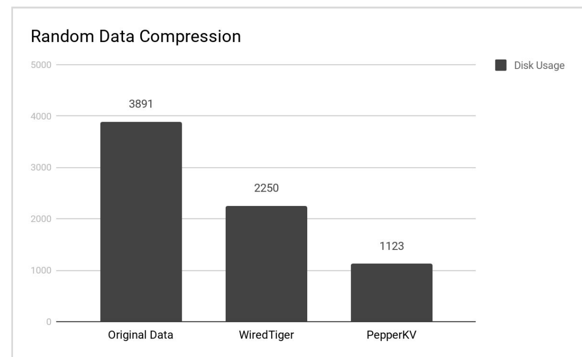

Figure 8. Compression Ratio on Alibaba Dataset

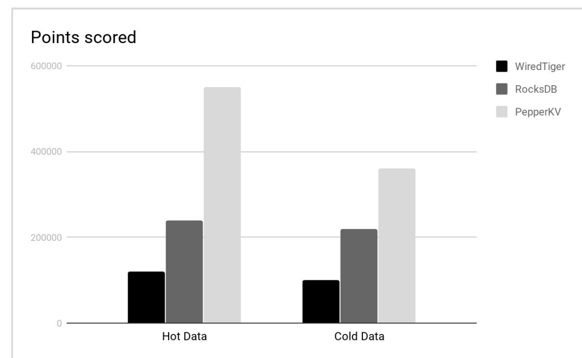

Figure 9. Random Reading on Alibaba Dataset

3) Test results from Baidu

• Comparison made with other engines that were highly optimized by Baidu

• Test scenario is not our optimal scenario

• TokuDB has been abandoned due to poor performance

• TerarkDB achieved the best compression ratio among all engines and our read performance is about 10 times better.

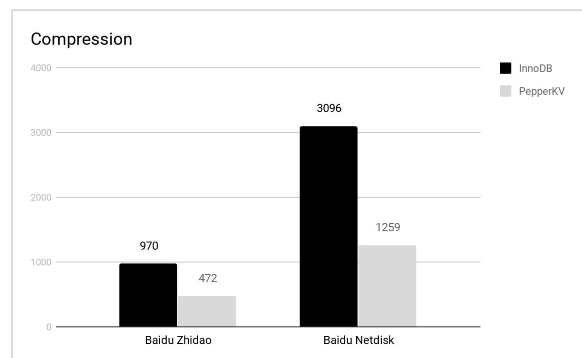

Figure 10. Compression Ratio on Baidu Dataset

2.3 DApp Store

Like Apple's App Store, we would love to integrate a DApp Store inside our wallet client. This DApp Store will help users download DApps directly from our DApp Storage Nodes and run it directly on their local machine. Here's the basic workflow of how to download and run a DApp:

1) Start your client side wallet, the wallet will then finds a set of fastest DApp storage nodes (e.g. 10 different nodes).
2) Enter the url / name of target DApp in your wallet's search box, the wallet will sends the query to all connected DApp storage nodes.
3) Combine the result of the responses from all DApp storage nodes.
4) Choose your target DApp and install it, after you install it, the wallet will automatically send the MD5 signature of the DApp to all other DApp storage nodes, and confirm that you are using the correct one.
5) If the DApp developer set a price for the DApp, you may need to pay for it to register your address.
6) Then the developer should pay the cost of the DApp storage.

As we see, this process is pretty straightforward, and the most important thing here is to keep DApp storage nodes professional, fast enough, secure and of course, profitable. We will setup our own DApp storage nodes at the beginning, and test the overall cost and income and make sure every participant here are happy. Relying our cutting-edge compression technology, we believe we can save the storage cost much better than others (3 ~ 10 times better), so it's a huge advantage.

2.4 Pepper Rank

Blockchain is very hard to upgrade, for example, if your platform migrates from PoW to PoS, the miner won't happy and may leads to a hard fork. This is because you are touching their benefit. We can't prevent anyone who want to do a hard fork, but what we can do is keep those most valuable users stay. So we introduce a new voting strategy based on user contribution, which defined by Pepper Rank. The main idea is, if someone important (has a higher rank) is transfer money to you, then your rank is increasing.

Link Structure of Addresses. Each address has a set of connections with other addresses, including send and receive transactions:

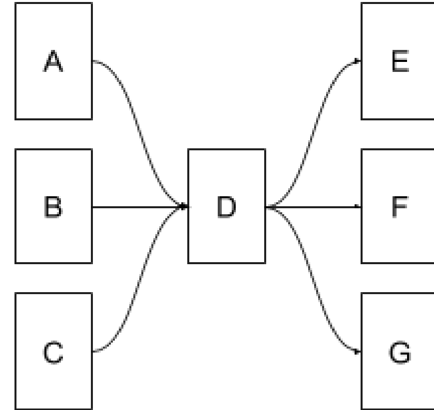

Figure 11. Address / Accounts Links

The definition of Pepper Rank. Let u be an account. Then let $F(u)$ be the set of accounts u points to (have transferred tokens to them) and $B(u)$ be the set of accounts that point to u (have received tokens from them). Let $N(u) = |F(u)|$ be the number of links from u and let c be a factor used for normalization (so that the total rank of all accounts is constant). We begin by dening a simple ranking, R which is a slightly simplified version of Pepper Rank:

$$R(u) = c \sum_{v \varepsilon B(u)} \frac{R(v)}{N_v}$$

Computing Pepper Rank. The computation of Pepper Rank is fairly straightforward if we ignore the issues of scale. Let S be almost any vector over accounts (for example E). Then Pepper Rank may be computed as follows:

$$R_0 \leftarrow S$$

loop :

$$R_{i+1} \leftarrow AR_i$$

$$d \leftarrow \|R_i\|_1 - \|R_{i+1}\|_1$$

$$R_{i+1} \leftarrow R_{i+1} + dE$$

$$\delta \leftarrow \|R_{i+1}\|_1 - \|R_i\|_1$$

while $\delta > \varepsilon$

Note that the d factor increases the rate of convergence and maintains $||R||_1$. An alternative normalization is to multiply R by the appropriate factor. The use of d may have a small impact on the in uence of E.

There are some critical problems here we need to deal with, especially ***Anti-Fraud.*** The most common way is to generate a lot of address, and give them some tokens and transfer these tokens back to one central account. But it actually doesn't work, because the connections from those new created accounts, are not important and their rank is so low that can almost do nothing to the central account's rank. Unlike traditional PageRank, it is even harder to cheat on this fully distributed token economy. And of course Anti-Fraud is a long term task for us.

2.5 Off-Chain Storage

We will not put off-chain storage online at our first few steps. But the basic idea is to use a proof-of-storage strategy for off-chain consensus, the architecture of this part is here:

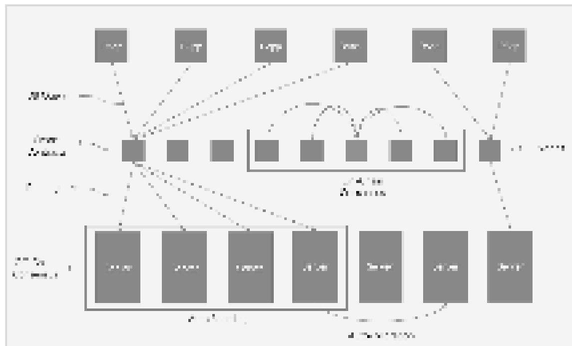

Figure 12. Off-Chain Storage Architecture

2.5.1 Off-Chain Consensus. For those on-chain data, PepperDB uses traditional methods like PoS or PoW (doesn't matter at the moment, both of them need to be improved in the future). But for other data which is not necessary to be stored on-chain, PepperDB saves them into off-chain data nodes.

Paxos protocol assumes all nodes are honest and none of them are fraudulent nodes. But this is not realistic in a blockchain environment. So, by modifying the Paxos protocol we can achieve byzantine fault tolerance. The basic idea is to add a new stage before Paxos's preparation stage – we call it pre-preparation stage (similar to PBFT, Practical Byzantine Fault Tolerance).

At the same time, PepperDB uses data node's deposit credit and account balance to define its weight. Data will be changed only after enough weighted nodes have approved the update. So, if any node wants to do something evil, the cost would be too huge and therefore none of the nodes would want to try it. And of course, we will have proof-of-storage to make sure each of the nodes are storing data properly.

2.5.2 Find Off-Chain Data Node. We are expecting all data nodes' storage service to be stable and predictable. So we suggest and encourage professional teams and hosting companies to host data nodes. Any other participants could also join the network as data nodes but it may not be as efficient, economically speaking. Most of the users in the network will be only Workers (Help to route queries to proper data nodes and get token reward).

By credit deposit and online rate reward, the network will be more and more healthy and strong (and remain decentralized).

When a developer wants to publish a DApp, he should first choose how many replicates he wants and estimate the daily data growth. The network will suggest him a proper network fee and estimated availability rate. Then after the developer confirms these conditions, the data cluster will be created.

After all data nodes are confirmed, everything will be written into the network as a smart contract, and then use an algorand-like algorithm to broadcast to the whole network.

Availability Rate calculation: Let's assume we have 10 data nodes, each of them has an online probability $P_i(i \in 1 \ldots 100)$. Users' data would be affected only when all replicates data nodes are down. The overall probability is $C_{10}^{1}(1-P_i)$, i is all data nodes that are storing the same data replicate. If we assume the average online rate is 60%, the whole dataset will have 99.98% online rate. Average online rate is 70%, then 99.98% in total and if average online rate is 80%, the result would be 99.99999%. Just like we mentioned, we encourage professional teams to host data nodes, so in most cases the single node online rate will be over 90%. And besides that, we will also set up official data nodes to make sure the whole system is working properly.

2.5.3 Online-Rate Calculation.

Since we assume all data nodes are not reliable, we designed an online-rate calculation algorithm to encourage the data nodes to remain online as much time as they can. When data is stored into a data node, each time the query cannot read data from the node, its online-rate will be affected. And its credit deposit will be taken when it hits the offline threshold. If a data node keeps being offline too often, the network will replicate its data into other servers and remove it from the network. Besides that, we have also designed a proof-of-storage mechanism to make sure all data nodes store data properly. This mechanism will automatically execute periodically.

2.5.4 Proof-of-Storage.

Data nodes need to be verified by proof-of-storage, in case some of the nodes delete everything after they get those data. Normally the simplest validation logic is: let all other parallel data nodes who are storing the same slice of data to sample the same bytes, then ask the target node to respond with the correct data. This is simple and effective but takes too much bandwidth. We will try to use a much better, lower cost way to achieve the same result:

1) When a worker decides to package a block, it can choose to take the responsibility of storage validation.
2) When any DApp needs proof-of-storage validation (triggered by smart contract), the current validating worker will pick a random generated integer and send it to the data cluster as validation number.
3) After receiving the validation number, the data node should sample some data from the start (use total data length mod validation number). Then encode it by MD5 and return the encoded hex chars.
4) Validators (e.g. the worker) will verify the result of all returned hex chars and mark those un-honest nodes.
5) Validators then pack all information, including the storage proof into a block and broadcast it to the whole network.
6) When any other workers receive the block, they will verify it more times, and only after a certain number of validators have confirmed it, this validation process will be marked as finished.
7) If any data node has a lower online-rate than the specified threshold, the network will kick it out of the system and punish it by confiscating its credit deposit.

3 TOKEN ALLOCATION AND ECONOMY

PepperDB will be use PDT (PepperDB Token) as its token symbol. The initial amount is 100 million and each year 5 million new tokens will be generated as community motivation cost and company expenses. PDT will be used in different ways:

1) Tokenize traditional internet products, build a strong connection with the existing centralize world.
2) Developers can use PDT to rent storage for DApp distribution and data storage.
3) Users could buy DApps directly inside their wallet and pay by PDT
4) Transaction fees.

Here's how we plan to use the initial PDT:

- 40% for fundraising
- 30% for the team
- 21% for previous investors
- 9% for community motivation, including airdrop, operations, developer plan etc.

REFERENCES

[1] Ankur Gupta, SUCCINCT DATA STRUCTURES, 2010

[2] J ´er ´emie Bourdon, Size and Path Length of Patricia Tries: Dynamical Sources Context, 2001

[3] Sergey Brin, Larry Page, The PageRank Citation Ranking: Bringing Order to the Web, 1998

[4] Nebulas: Decentralized Search Framework, 2018

[5] Satoshi Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System, 2009