Knowledge Coin

Stan Srednyak

October 3, 2020

Abstract

This paper discusses the conceptual design of a distributed search engine. At the core of this design are distributed storage, index and ranking systems. We demonstrate feasibility of such system. We show that it is possible to build a distributed advertisement service that generate revenue in analogy with the usual advertisement service. We show how to build a consensus mechanism that would distribute this revenue among the participating nodes. We examine a class of attacks on the system and demonstrate defense mechanisms that render the system insensitive to these attacks, as long as there are enough honest nodes in the system. We also discuss implications that existence of such system would have on the way search is conducted. We propose a conceptual design of a search bot ecosystem that can be useful in personal and enterprise information consumption and production. Most importantly, we show how to construct a market of open source algorithms that can be used by the ranking system and demonstrate how to build consensus mechanism that would allow algorithm developers get a share of advertisement revenue. We hope that our design, once implemented, will facilitate construction of more transparent , efficient and interactive knowledge system.

Contents

1 Introduction

It is very desirable to develop a decentralized search engine. Presently , commercial search engines are performing adequately at delivering high quality ranks. One has reasons to expect more in terms of summarization and dialogue systems for web content. Only web pages are delivered, and it seems far from possible to use these results to generate human level dialogue systems from these results. More importantly, the algorithms being used for ranking web pages are kept secret. They are probably the most guarded secret of the companies that provide search service. There has been long tradition to maintain open source software, and this tradition lead to the development of Linux OS and numerous open source packages. However, web search is an exception. The reason lies probably in the fact that it is very expensive to build a search system - web is very large, and it is beyond the scope of individual programmers or small companies to maintain such systems. However, in the past few years there were introduced new models of decentralized organizations based on blockchain technology [1, **?**, 2]. The revolutionary advantages of such new forms of organization are well recognized by existing organizations, such as

banks and governments. It is the purpose of this publication to show how to use this new form of organization to build a decentralized search engine.

The basic observation that underlies our construction stems from the well known organization of a search engine. A typical search engine is responsible for the following operations: crawling pages and maintaining a fresh local copy of the web, computation of the rank of pages, construction of the index for fast retrieval of search results, and various auxiliary services like query preprocessing, boolean manipulation of ranked lists based on logic of the query etc. It is the first three operations - crawl and storage , rank computation, indexing - that must match the scale of the web. Also, the ranked list retrieval must have low latency (1s at most) and the front end must have throughput which is easy to scale with the number of requests. We will show how to put all these operations on a network of nodes - computers that would perform a list of simple standardized operations, such that when they are allowed to communicate according to a protocol that we will discuss, the whole network will function as a search engine, satisfying the latency and throughput conditions we just mentioned. The nodes we have in mind are quite analogous to nodes running Ethereum software, but with more complicated set of functions that they have to perform and with more complicated ledger system (somewhat analogous to the one found in Hyperledger Fabric [1]).

We envisage our system as being similar to the Ethereum network, but with each node carrying part of the recent local copy of the web, part of the distributed hierarchical index, and ranking software, which will provide rank of the pages stored by the node. How would our system handle latency requirement? The answer lies at the structure of the index. As we explain in the following sections, is is possible to construct index in such a way that its depth is logarithmic in the web size. This index has a tree structure. The number of communications between nodes that are necessary for serving a query scales like $log_D(N)$, where D is certain constant (no less than 100, as we explain below), and N is the web size. We see that even for realistic size $N \sim 10^{12}$ of pages, this gives less than 10 consecutive calls. However, the network size for maintanance of such a system must be quite large. For the web size $N \sim 10^{10}$ pages, the network must have no less than $\sim 10^4$ standard nodes, as we show below. Of course, this is quite unrealistic for cryptocurrency networks - it is known that the whole Ethereum mining network is about 6000 nodes worldwide.

How do we overcome this problem? The answer lies in the nature of the revenue collected by companies that operate commercial search engines. What drives search engine business is advertisement. The latter creates massive inflow of funds to the search engine service providers. We demonstrate that it is possible to harness this revenue flow and direct it to the individual node maintainers, i.e., in the system that we propose advertisers will of course still be able to advertise, and the fees that they pay (in the usual bid mechanism type of billing) will go to the pockets of the network maintainers - the people who actually run the software, store pages, and deliver rank results to the end users. With this massive flow of income, we hope that individual people will be incentivized to invest in buying the necessary hardware to maintain parts of the distributed search engine at their homes, just as they are incentivized right now to buy the expensive mining hardware to keep mining rigs at their homes.

If there is one thing that we can be sure of at the moment, it is that the system we aspire to build will be subject to all possible attacks by very highly skilled programmers. We are most certain that they will densely explore the space of all possible attacks at all the joints of our system. Therefore, of paramount importance is to design protection mechanism that will make the system stable to such attacks. In fact, this requirement drives the design of our system. We build it in such a way as to make it attack resistant. The basic mechanism we use is duplication of parts of computation. Various pieces of computation will be duplicated by many network nodes. We build in a consensus mechanism that requires many nodes to agree on the computation before the results are being served to the user or before ads are delivered. One other crucial requirement is that the nodes participating in the calculation must be selected randomly - this is necessary to prevent adversaries from manipulating local parts of the computation, such as the calculation of ranks of individual sites. We propose a mechanism for achieving this. It is based on an element which we call Randomness Servers. These are web servers that are available with low latency and which can generate large quantities of high quality random numbers, and which are trusted. Thus, we wish to emphasize, our system is not entirely autonomous - these servers must be maintained by trusted individuals. This is the only external trust mechanism in our system. Fortunately, maintaining such servers is relatively inexpensive, and we plan to maintain some of them in the course of development.

Before we plunge into the details of our implementation, we wish to mention briefly the advantages that the system we planning to construct will give to the end users. We are talking about a knowledge system of a planetary scale. Internet is such a system. It is not perfect yet, and much can be done. We hope that the tools we will be building will serve this goal. Most notably, the search algorithms are not open source these days. We will make them open. In fact, we will create algorithms market that will allow developers to systematically research new algorithms and fine tune the old ones. The successful algorithms will have the opportunity to be adapted by the distributed search engine and will generate revenue to their developers (through part of the advertisement fees). We hope that this will foster wide scale research into ranking algorithms, text summarization and knowledge modeling (so far , the academia was basically excluded from this research because it is impossible to do without access to a web scale data, this kind of data is not available to the academics). There has been a lot of discussion about the ways to construct "knowledge graphs", and such systems were indeed constructed in some scientific domains (like Elsevier journal system). We hope to provide the necessary data structures to allow for systematic

research in this direction.

We should mention at this point the general nature of search process. Search process is a dialogue a person has with world knowledge system. This dialogue is very valuable. It concerns the very quest for new knowledge. The way people interact with the knowledge system should be treated with great care. This information should be carefully accumulated so that people can use it (of course, there are multiple privacy concerns here, and we discuss them to some detail in appropriate sections). We think that the stream of web searches from a person constitutes a kind of digital reflection of personality, or web reflection. This is true also about larger organizations, like labs or industrial companies. Therefore, we will invest efforts into development of search bots - a software suite that will allow individuals and organizations to manage their individual knowledge graphs that they collected and constructed in their interaction with the world repository of knowledge. Of course, this ambitious goal will require development of specialized interaction mechanisms and ledger systems. We found it convenient to introduce special cryptocurrency - knowledge coin - to allow people, organizations and bots they create to manage differential privacy settings and data communication protocols. We think that with the development of the knowledge system there will emerge a kind of knowledge market. We should note that for the purposes of managing advertisement and other internal mechanisms, we introduce several other coins (rank, index, and advertisement coins), and these are different from the knowledge coin.

The plan of the paper is the following. After review of some relevant literature, we describe the architecture of our system. Then we discibe mechanisms of defense against attacks, privacy model, ledgers and consensus mechanisms, our vision of bot system, knowledge system , and our proposed timeline.

2 Related work

Although the recent blockchain revolution brought about many attempts at decentralized finance, usage of blockchains, sharded ledgers DAGs etc. in organization of decentralized organizations, there were relatively few attempts to decentralize search, for notable exceptions see []. This is despite the fact that desirability of decentralization of search is well acknowledged by the community. This is even more surprising since the architecture of a search engine is such that it is very amenable for distribution over a network of nodes, as we will see in the following section.

2.1 Internet of Things

In recent years there emerged huge literature on so-called Internet of Things (see e.g. books [3, 4]). We hope to contribute to these developments by building world's knowledge system. The world wide web is a good knowledge system but it can be improved in many aspects, in particular , in the way inferences are made and the way humans and organizations interact with it. For this purpose, we will develop the market of search, rank, and summarization algorithms, and the search bot ecosystem.

2.2 Hyperledger Fabric

We owe to the Hyperledger project [1] the idea of organizing the interaction of multiple parts of our system in terms of ledgers with multiple permission types. Decentralized search can only be made possible with comprehensive rules of transaction records.

2.3 DAG and blockchains

There has been interesting research concerning alternatives to blockchains - DAGs of various sorts [5, 6]. It is easy to see that the database of transactions in our system will be huge. Therefore, out of necessity , we will maintain it in a distributed form. However, due to our finalization of transactions system, we do not have the need to maintain this database forever. We will have a retirement mechanism for old data.

2.4 Open social networks

There are many posts on social media that discuss and advocate decentralization of social platforms. Many such systems were realized [7, ?, ?]. In the context of search, we think of "search bots" as web representatives of people. We plan to organize certain "social platform" (or ecosystem) for these entities, where there will be ample opportunities to program, teach and deploy artificial systems of various complexity.

2.5 Zcash and crypto

There emerged revolutionary approaches to maintenance of security and privacy in classical financial transactions [8]. In our system, we did not find applications for these, because of the specificity of privacy requirements: there

is no monetary flow from searchers, and there is only one-way monetary flow in the advertisement system. For these purposes, a simple privacy model is sufficient (see section on privacy).

2.6 Tezos

The Tezos project [9] proposed to use evolving consensus algorithms. This is where we first learned about this idea. However, it seems it has become the mainstream, especially after announcement of EOS .

2.7 Filecoin

Our market mechanism is directly inspired by that of filecoin [10]. We think that the idea of having a market for storage and having a specific coin for its functioning is revolutionary. In our situation,, the task is somewhat simpler - we do not need to guarantee storage, we just need to guarantee rank computation, therefore, we do not need the intricate Proof of Storage mechanism described in that paper. We do not propose to adapt IPFS for storage in our system, because that technology is still unstable, but we plan to maintain support for operation on IPFS. We also plan to explore possibility of adaptation of libp2p [11] to our system. Perhaps, it can be speed up the operation.

3 Architecture

There are several basic account types in the system. First, there are accounts of search bots (s-bots for short), in particular, people. If the person does not wish to store his search history, he does not have to register or sign up. A one-time account will be generated by the system. However, if the person wishes to save history, or write a script that will constantly interact with the system, continuously querying it, storing and accumulating information, there will be such possibility. This history information can be made part of the search and ranking procedures. Simplest s-bots are basically html pages loaded by a browser.

Next, there are bots that manage advertising accounts, in particular, human advertisers. These accounts must be linked to a wallet that contains corresponding set of coins (for advertising, only one type of coins will be needed, namely, ad-coin). The theory of digital wallets is well advanced nowadays, so we will not go into details here.

Other account types are structured according to the function they will carry out. These are essentially IDs of machines and servers, although it will require some modest human input (like running a script) to activate them. There are crawl bots, document server and ranking bots, index bots and brokers. In addition, there may be added verifier bots, if we see the need for this. Document server bots will be denoted by DS for brevity. Storage and ranking will not be separated in our system. DS and index bots (or i-bots) will constitute the majority in the architecture of the system. Brokers serve the role of the front end. They are responsible for knowing the list of local indexes , to which they send the queries once obtained from s-bots. They are also responsible for scanning the local advertisement ledger, locating the winners of the bidding auction for ads, and serving the adwords to users. Consensus among randomly selected brokers is the mechanism that finalizes ad service transactions.

We hope that the role of other bot types will be self evident in the following sections. We wish to emphasize that each bot will need to have its own wallet to operate. There will be several types of coins, corresponding to the types of functions performed by bots. Thus, there will be crawl, rank, index and advertisement coins. They will be freely tradable on the market.

One more element are the Randomness Servers. This is very important element that provides security to the system. It ensures that if the majority of the nodes are honest, the system will serve the correct rank. We will analyze in detail attacks on various subsystems and junctions.

3.1 Index Tree and Return Tree

The basic architecture of a search engine data center is described in the official Google publications [12, 13] (with notable exception of the load balancer and local rank combination mechanisms). This architecture is amenable to decentralization, as we explain in this section. We envisage the key components of the system to be spread out among a set of nodes (network). The key component that makes fast search possible is the index. The index has the following basic structure

$$I[q] = [DS_1, ..., DS_n] \tag{1}$$

where q is a keyword and DS_i are identifiers for the Document Servers. This is the complete list of DSs that contain the given word, together with the DocID information and location on the disk (this is necessary for fast search). When a query arrives at the front end (or Broker, how we will denote it for brevity), the broker B will issue a command of rank calculation to the DSs from this list. This list may be quite large - beyond capacity of

a single machine. It is therefore necessary to distribute it. This can be done as follows. Index can be organized in layers $I(n)[q]$, where the number n denotes layer. Only the lists of layer 0 contain addresses of DSs

$$I(0)[q] = [DS_1, ..., DS_d] \tag{2}$$

Indexes of higher level n contain addresses of servers that contain indexes of level $n-1$

$$I(n)[q] = [I_1(n-1)[q], ..., I_d(n-1)[q]] \tag{3}$$

We will call this structure a hierarchical index. For simplicity, we can envisage it as d-ary tree. There must be a mechanism that keeps this tree well balanced, so that it has approximately equal branching number.

The height of this tree is $log_d(N)$, where N is the number of the pages that contain the given word q. It cannot exceed the number of pages on the web. This logarithmic scaling is the key to the feasibility of the systems: although the index is distributed, the number of consecutive URL requests is only logarithmic in web size, which gives us hope that the necessary low latency can indeed be achieved. For example, if t is an average time it takes to transmit information between nodes in the network, then the total time for search is

$$T = t \times log_d(N) \tag{4}$$

We immediately recognize the need to keep operations local. It is inevitable that for low latency there must be many indexes , distributed geographically. They are the analogies of the Google data centers, which have strong geographic localization.

3.2 Return Tree

There ranking information , together with web page addresses and snippets of information, should be returned to the broker B. It is unfeasible and unnecessary to return the list of all pages that contain q. Google search engine does not return this list. It only returns the top of the list, say $M = 1000$ pages. We will copy this strategy and show how to decentralize this step in the computation. In a sense, this construction mirrors the structure of the Index, described above. Apart from DSs, we will need special Transmission servers (which may be implemented by installing additional piece of software on the standard node). These transmitter nodes are organized in levels, with only transmitters of the level 0 being in direct contact with DSs. The transmitters of level n are in contact with d transmitters of level $n-1$ and they receive lists of pairs (d_i, r_i) of ranking information of length at most M. In total, they receive up to

$$M * d \tag{5}$$

rank pairs. The transmitter node then merges these lists and takes the top M of entries and submits this list to the transmitter at level $n+1$. Again, we obtain an approximately regular tree of degree d.

3.3 On the number of leaf nodes

We should point out that the number of leaf nodes can be quite large. The tree of index exists to minimize the latency associated with network transmission. The number d must be chosen in such a way as to fit the bandwidth capacity of the node. In other words, the number

$$d \times size\_of(Q) \tag{6}$$

(the information transmitted down the index tree by a typical node) should fit the bandwidth, Q being the query size (which must include the address of the broker node, in case return tree is different from index tree). It therefore does not make sense to put several pieces of $I(0)[q]$ on the same server, for the same q (although it is entirely possible for different q). This suggests the lower number of nodes in the system to be

$$N_0/d \tag{7}$$

where N_0 is the maximal number of pages of interest. We emphasize at this moment that index should not contain "stop words" (common words). It should be limited to the tail of the Zipf's law, i.e. the words that carry actual information. A conservative estimate then would be $N_0 \sim 10^7$ (10 million pages). With $d \sim 1000$ this gives about 10^4 servers.

3.4 Rank computation

Computation of a good rank is at the core of a successful search engine. In principle, the rank can be very complicated function. It must include hyperlink information, semantic page information, javascript code information and many other things like reputation scores of the author (if it is an article), etc. It is impossible to predict at the moment what the total list of factors will eventually become. An approach based on "learning to rank" [14]

should be adapted. We discuss here only the simplest case - the computation of PageRank. We think that the following property nonetheless must hold for arbitrary rank systems:

$$Rank = f(Network\ rank,\ local\ rank\ based\ on\ Q) \tag{8}$$

where network rank is certain pre-computed vector, that is independent on query, and local rank based on Q is a vector that contains semantic information about the page and its relationship to the query Q, which is computable based on the page information alone. This local rank contains information like position of the words in Q inside the page and sentiment analysis of the page. In principle, this local rank function should be trained using global data (it can be a kind of neural network [15]) , but its evaluation should be local to the page.

In the case of network rank, it can be computed iteratively. In practice, the network rank should be updated continuously, but for simplicity of the exposition, we will assume that this rank is updated at discrete time episodes, at the end of which the DSs must come to a consensus regarding the rank. This may be implemented by maintaining a distributed tree of rank transactions (DTRank for short). We chose a tree architecture for the sake of scalability. We envisage branches of the tree to be local, although the is not necessary: in the course of evolution they can migrate geographically. The ranking transactions do not contain valuable data, once new rank is calculated , and we chose not to chain them, making them disposable after certain period of time, once the consistency of the rank computation is checked.

In an iteration of network rank computation, a DS receives (the vectors of) network ranks of all the pages that link to this page and uses the update rule

$$r' = f(r_1, ..., r_n) \tag{9}$$

(this is analogous to PageRank, but the function may be more complex , to accommodate semantic information). At the end of the iterations, all the DSs in the given locality , that contain this particular page d_i, must have very close ranks $r_1, ..., r_s$. We will implement smart contract on the DTRank in such a way , that it will record a ranking transaction (RTX for short) if and only if the given majority p (say, 90%) of DSs agrees in their rank to the precision ϵ. Otherwise, transaction is rejected. The transaction will have structure

$$RTX =< t,\ [DS_1, ..., DS_n],\ IndexID,\ pageID,\ rank > \tag{10}$$

IndexID is necessary here because it has to be used in order to find those DSs that contain the page. Note that this transaction tree can grow very large. We can use compressed transaction tree with transaction structure

$$RTX =< t, DS_1, ..., DS_m, IndexID, [d_i, r_i] > \tag{11}$$

Now, the transaction includes the list of DSs from given locality, index used, and the list of network ranks computed by DSs (on which they have to agree to precision ϵ). This transaction information is quite large and must be stored in the distributed fashion. Fortunately, in contrast to the usual blockchains like bitcoin, this information does not have to be stored forever, and can be discarded after the payments have been processed and verified (see below) and all the necessary network maintenance jobs have been done. We therefore think of the rank transaction data base as temporary structure, necessary to reach local consensus, ensure consistency of rank computation, and distribute fairly the revenues from advertising. However, it is very different from the ledger of transactions in cryptocurrencies in that is does not have to protect against the double spending problem. It is more similar to the architecture proposed by Hyperledger [1], and adapted to the search engine architecture.

We think of the transactions of the above type to contain digital signatures of the servers involved. This information is useful for keeping track of the malicious or malfunctioning servers.

3.5 Serving the Rank

It is not enough to accurately evaluate the network rank, it is also necessary to deliver the correct combination of the network rank and local semantic rank to the broker. This is a potential attack place for hackers. It is quite tempting for a hacker to pretend that, for example, it is an honest T-node of level 0 and then manipulate the ranks submitted to him by the DSs. In this subsection we propose a protection mechanism against such kind of attack. The basic mechanism is quite straightforward. We start at level 0. How do we ensure that DSs that send ranks to the T-node of level 0 provide honest rank?

This check must be done by the 0-level T-node. Suppose that the majority of local DSs that serve rank are honest (how to achieve this is discussed in the section on crawling and data distribution among DSs; at this point we also must fight the situation when malicious DSs flock on a particular page with the purpose of manipulating its rank. We propose a mechanism to defend against this in the section on crawling). Suppose that the 0-level T-node is also honest (the situation when a 0-level T-node cheats is considered below). Then all that has to be done is to allow the T-node to poll the population of DSs randomly and take the majority vote. If there are not many local DSs that serve this particular page, this is easy. But in the ideal situation, a randomization mechanism must be used. We chose to rely on yet another element of our system: Randomness Servers (RS). The

T-node , upon receipt of the rank from its pool of DSs must send a request to an RS , that will return a set of duplicating DSs $DS_1^k, ..., DS_n^k$ that contain the same pages (at this stage it becomes clear why we must choose our DS duplication mechanism in block fashion - if we decided to allow disordered duplication, then the RS must return a list of duplicating servers , rather than duplicating blocks, and this will require too much data transfer between the RS and the T-node). RS node signs the list of the random blocks with its signature. Then T-node polls the DSs from this random list for the rank. If the majority of the DSs agree on the rank, this part of the Rank Service Transaction is finalized. It contains the following information

$$RSTX =< t,\ T\_ID,\ RS\_ID,\ RS\_ID\ signature,\ [DS_1, ..., DS_n], [DS_1^1, ..., DS_n^1], ..., [DS_1^k, ..., DS_n^k],\ (d_i, r_i) > \tag{12}$$

The transaction is finalized by the RS who checks that the list of duplicating DSs coincides with the one that it produces. Thus, RS is external to our system. This is the only element of the system that involves trust.

Now we can move on to the consideration of cheating T-node of level 0, again assuming that the majority of DSs are honest. But then we can employ the same mechanism as above, only instead of DSs we will have a block of T-nodes of level 0. In this way, T-node of level 1 verifies the integrity of computation performed on its branch. This apparently may lead to exponential blow up of the computation necessary - because each of the T-nodes from the previous layer must ask its own set of nodes of lower level to confirm the computation. We propose the following solution to this problem. The only purpose to invoke the set of level 0 T-nodes $T_1, ..., T_n$ is to check the given node T. They are randomly selected and are not known to each other and the node T (they are known only to RS, which originated this list and to $T^{(2)}$ that conducts the check, which must maintain a secure communication channel). We may use this channel to communicate to $T^{(2)}$ the lists $[DS_1, ..., DS_n], [DS_1^1, ..., DS_n^1], [DS_1^k, ..., DS_n^k]$ of the DSs that were used in the computation. Then $T^{(2)}$ can request these DSs to send their informations to $T_1, ..., T_n$ (again, on the network level it must be ensured that $T_1, ..., T_n$ cannot "eavesdrop" on the network traffic from $T^{(2)}$. This can be achieved by encoding using public key cryptography and randomly sending more fake signals to the larger population of DSs, to confuse the eavesdroppers. Alternatively, the communication to DSs can be relegated to the RS, $T^{(2)}$ can be excluded from the process and no assumption of its honesty must be made. In fact, this information does not need to be transferred to RS either - it can be coded and stored on a particular node, and only this location should be transferred to RS, thus reducing the amount of information transferred. The RS then sends a request to a randomly selected index of level 0, that points to the same set of DSs, and in this request point to the place where the list of DSs is stored. Thus, the whole load of information transmitted is a constant, independent of the size of the list of DSs). Thus, the same set of DSs can be used. There is only slight overhead of sending information to larger set of T-nodes. In this way, the amount of computation increases only linearly, rather than exponentially.

The total Rank Serving transaction therefore has a rather complicated structure. In fact, it is a tree of elementary transactions, performed at each layer of the return tree

$$RTTX(n) =< t, T(n),\ , [T_1(n-1), ..., T_k(n-1)], [T_1^s(n-1), ..., T_k^s(n-1)], \{RTTX_i(n-1)\} > \tag{13}$$

3.6 Crawling

Crawlers will serve several purposes in our architecture. Traditionally, a crawler is subdivided into several structure blocks (DNS resolution server, fetcher, parser etc.). We will have to add to this list also reallocators and index updaters. Our crawled page will have more structure: in addition to the usual html , it will have a summary page , that contains the last update time and various statistics (e.g. ranks) and most importantly usage information. First problem that we have to address is how to ensure that a page is covered by DSs the majority of which are honest? This has to be decided at crawl and storage stage. The solution that we propose is described below.

First, we do not allow DSs to make decisions on what to store. This is decided by Randomness Servers, that ensure that it is impossible to predict at which DS a newly crawled page will be stored. Suppose a new page is crawled. Then, according to the smart contract on the Ledger of Crawl Transactions, to add a valid transaction, the crawler must contact an RS, which would choose randomly a DS from the pool of available DSs in the given geolocation (there may be several DSs chosen, as we will have a need for duplication of pages, in case of failures; they all must be random). The DS then receives the page for further analysis and storage. The Crawl and Storage transaction then has the following structure

$$CSTX =< t,\ pageID,\ CrawlerID,\ RS\_ID,\ RS\_signature,\ DS_1, ..., DS_n > \tag{14}$$

It will also be the responsibility of the Crawler to check if a DS holds a particular page for long time, it deletes it, crawls it again and stores at a newly randomly chosen docserver. This will ensure that DSs will not be "corrupted" to manipulate the rank of a specific page.

Crawler is also responsible for making updates of existing indexes. If a new page is crawled, the set of key words is parsed. For each of these keywords, appropriate index is consulted and the DS_{ID} is propagated to the bottom of the hierarchical index, where is it added to the list. If the list becomes too large, it can split in two.

Crawlers also make sure that the coverage of a page by DSs is proportional to its popularity: this is parallelization mechanism that will ensure that there are enough DSs to serve the rank for highly popular pages. Therefore,

for local copy of the web, there will be maintained a data structure that records the number of DSs that serve particular page (this can be quickly computed from index).

3.7 Index update

To ensure computational integrity, at each geolocation several indices will be maintained, which duplicate and check each other at rank delivery stage, as we described above.

The coverage of particular page by servers in a particular geospatial location should be sufficient for cross validation of their rank. This requirement will be built in the chaincode of rank construction ledger.

3.8 Advetisement system

The mechanism that drives search business is the desire of companies to put their advertisements on the search results page. We therefore need to create a system that will allow for the distribution of this monetary investment among the participants, the subsystems that were involved in the creation of search result (crawlers, document servers, indexers). Although there may be many solutions, we describe one of them in this subsection.

The first element is the Advertisement Market (AM). AM can be realized by a sharded tree of transactions, sliced by time (i.e., transactions are finalized in determined time interval). The advertisers (ad-bots in the following) bid on keywords, geospatial location, and possibly other variables (e.g., history of the the search bot , to which the ad is being served, if such history is available to the search engine). The bidding transaction is a smart contract that locks the specified amount of money , which will be deducted from the account of ad-bot , if its advertisement is served and consumed (see more on verified consumption mechanism below). It is the responsibility of the Broker node that served the ads to finalize this Advertisement Transaction by providing the evidence that it delivered the ads. It seems to us that a special currency should be introduced to serve this advertisement market. We will call it adcoin (advertisement coin). We will create a market at which adbots can purchase this currency at market price to initiate the bidding transaction. The bidding transaction therefore contains the following data

$$BTX =< t,\ query\ words,\ location,\ adbot\ ID,\ adbot\ signature,\ bid\ price > \tag{15}$$

It is initiated by a adbot that has sufficient funds. It is finalized by a broker that supplies the following information

$$BTX_1 =< t, brokerID, [broker_1, ..., broker_s], RS\ ID, RS\ signature, search\ bot\ ID > \tag{16}$$

Here again we require the broker to contact a randomness server that selects randomly a set of nearby brokers $[broker_1, ..., broker_s]$ to which the click information from the browser will be sent. These brokers will have to sign the transaction as well.

We have to address the situation when there are malicious brokers, that just do not want any transactions to happen, that will reject any transaction, or have random behavior. We propose a majority vote mechanism. The chaincode just counts the votes and if more than $p\%$ brokers confirm the ad consumption, the transaction is finalized.

On another extreme, brokers apparently do not have any incentive to check the click information. They can just let fraudulent transactions to pass. Therefore, we will require them to submit the encrypted click pattern that they received from the browser. These brokers must be kept unknown to each other, to prevent collusion. It is the responsibility of the randomness server to interact with the browser (JS code of the search page, to be more precise) of the search bot and provide it with the IP addresses of selected brokers to which to send the duplicate click information, which these brokers will next submit to the blockchain. Although this is a challenging networking problem, we think the modern technology provides off-the-shelf solutions for this problem [16]. If the click information from these brokers does not match, the ad consumption transaction is rejected.

Finally, we should mention that there is possibility to emulate clicking behavior of an honest human subject , to ramp up apparent ad consumption numbers. This is well known "ad fraud" problem, and there is extensive literature on this subject , see e.g. [16]. A combination of these solutions can be used to counteract this fraudulent behavior.

The distribution of the adcoins collected from the adbot will be described in the next section.

3.9 The Market System

We propose a market mechanism of organization of the service of crawl , storage, index and rank.

3.9.1 Crawl market

It is the responsibility of DSs to maintain up-to-date state of the pages. Each page has a document page attached to it which contains details of the rank calculation and the time when this rank was updated, and the time the

page was last updated by the crawler system. We can call this Storage Transaction

$$STX =< t, crawler\ signature,\ [DS_i\ signature] > \tag{17}$$

where DS_i are the IDs of DS that verified the rank. If the time lag is too large, the probability of the broker to reject this transaction should increase. We will implement a chain code for this purpose. The update mechanism is the following. DS looks at the Crawler Market and finds a set of crawlers that can fetch the page. DS then initiates the crawl transaction and also sends a request to RS to get a random subset of crawlers from his list to whom to assign the crawl job. RS returns the list. Then DS sends transaction request to the crawlers $C_1, ..., C_n$. After they return pages, the DS sees if the fraction p of crawlers agree on contents and stores the page (and initiates next steps that are necessary for update of the rank and index). It also finalizes the transaction

$$CTX =< t,\ docID,\ [C_1, ..., C_n],\ RS\ signature,\ X_i > \tag{18}$$

X is the sum of crawl coin that DS pays to the crawler C_i. In our design, there is special currency that serves crawl market. It is exchangeable to other currencies within the system. Note that details of the implementation of the crawler are never part of this transaction. In this way , the system becomes flexible: crawlers are free to implement their own routines to crawl pages (and do other necessary operations like DNS resolution). We think that this mechanism will foster efficiency. It will also be important to maintain a standard set of crawlers on the GitHub, so that they can be used as plug-and-play.

3.9.2 Storage and Rank markets

Storage and Ranking will be done by the same servers, as the ranking process is inseparable from data. Therefore, we will have only Rank Market. On this market, there is maintained a list of local DSs that can serve docIDs and their related information (ideally, detailed semantic analysis of the pages). A broker that wishes to serve a search request then selects a set of Indexes, each of which has proven that it covers the web (see next section). The mechanism of index tree and return tree operation was discussed to some detail above. Here we note that the Index (the whole tree, represented by the root node $I(d)[q]$) functions as a single unit. The distributed tree of storage transactions should be maintained by the system that contains information about the documents and DocIDs they store.

When indexer requests rank from a DS that it contains, it initiates a transaction of Rank Request. It has to submit rank request to a set of DSs that contain the same page, as we discussed above. It is these transactions that are recorded in the distributed database

$$RRTX =< t,\ docID,\ [DS_1, ..., DS_n],\ I(0)[q]\ ID,\ RS\ signature,\ price > \tag{19}$$

This transaction must be signed and stored at DSs. It has to be kept long enough to finalize distribution of payment to all participants. These transactions will have retirement age as they are needed to maintain record of system malfunction and fair implementation of chaincode that is responsible for distribution of revenue.

3.9.3 Index Market

We choose to implement the indexing as a market for efficiency. The transaction on this distributed tree is initiated by a broker that selects a set of indexes to collect document rank. The transaction has simple form

$$IRTX =< t,\ q,\ B,\ [I_1, ..., I_n],\ RS,\ P > \tag{20}$$

Here B is the broker, $[I_1, ..., I_n]$ is the set of indexes selected randomly from the pool compiled by the broker and sent to the randomness server. P is the price on which index and broker agreed for the index to deliver the rank to the broker. This price is set in index coins , or i-coins, which can be purchased at the exchange (we remind that we will implement these exchanges as internal to the system; the auxiliary coins should be internal to the system).

3.9.4 Distribution of revenue system

Distribution of revenue will be implemented by the chaincode on the advertisement ledger. Once transaction is finalized, and ad consumption is confirmed according to the consensus protocol, the chain code will distribute the money collected from adbot in proportion to all the actors that participated in the service of the transaction - DSs, indexes, and the broker. The proportion is subject to change and should be dictated by a market mechanism. There are at least two implementations.

One of them carries the advantage of locality. In this implementation, all the revenue from advertisement is collected solely by the broker. Thus the funds transfer is localized at the ad ledger. This method relies on the market mechanism to transfer funds to other layers (indexes, DSs, and Crawlers). In essence, for the system to function, broker must pay a fair price to the indexes and return trees to deliver the rank, and the indexes must

pay a fair price to the DSs in order to get the rank from them. Note that in this mechanism , the Rank Request transactions are not part of the payment system that transfers funds from adbot to the broker.

Another mechanism can involves the totality of operations and corresponding actors and their accounts. The chaincode on the ad ledger distributes the funds to all the actor involved, including DSs. Crawlers get their revenue differently - they are not immediately part of rank service transaction.

4 Attacks

The typical operation of the system consists of the following stages:

A) A DS initiates a crawl transaction. A crawler fetches the page and submits it to the DS. DS checks if the links on the page are represented in the local DS network and updates the page info.

B) Rank is computed recursively by communication among DSs that submit rank functionals to the pages given docID points to.

C) Index is updated.

When a query arrives to a broker the following chain of transactions takes place:

1) Broker selects an index at the Index Market and passes the request down that index.

2) The index then issues rank request transactions to DSs from the 0 level leaves of the index.

3) DSs return the ranks to the index at level 0.

4) The ranks are merged and propagated up the index tree and eventually returned to the broker.

5) Broker selects the winning ad bid at the Ad Ledger and returns the top M pages together with ads to the user.

Lets consider to some detail attacks on this system. In principle, all stages in the operation will be subject to all possible attacks. Our basic strategy to defend against these attacks revolves around three principles:

a) duplicate each step. Make several actors of the same capacity perform it and cross-check their results. Transaction takes place if and only if majority confirms the correctness of the result. In principle, bad actors are not punished - nodes can behave as badly as they wish. If the majority of the nodes involved in particular transaction are honest, the transaction will be confirmed.

b) use a randomness mechanism to prevent bad actors to concentrate in one particular place of the system to corrupt it. This is essentially external element of the system and it involves a trust mechanism. However, maintenance of this external system of Randomness Servers is cheap - these are just uniformly distributed servers that can serve large quantities of good random numbers. There is no expensive infrastructure involved here.

c) to defend against adversarial behavior at the tree structures (index and return tree) , perform checks by layer. First make sure that transactions at given layer are valid, then proceed to the next. It involves direct communication of layers n and $n+2$. This communication can be organized in a way that servers from layer $n+1$ are kept ignorant of the choices made in this communication. Assuming that the results at layer n are correct, the checking procedure is organized in such a way that honesty of the $n+2$ layer servers is not required. It is still possible to check the servers at layer $n+1$ even with malicious servers at layer $n+2$ (as long as the majority of servers at layer $n+1$ are honest).

5 Privacy

There are two kinds of actors in our system for which privacy may be important - search bots and ad bots. For the first we will organize a differential privacy instrument - search history market. As we explain in the section on bot ecosystem, bots should be allowed to trade knowledge they gained from the system. As we will explain there will be maintained a ledger of queries. But these queries will be anonymized - the botID will be hashed with time appended to it. Keywords will be visible and available for analysis.

Concerning the ad bots, privacy guarantees will be stronger. The contents served by the broker will be encrypted, and direct communication between the ad bot and the browser of the user will be required to unlock the contents of the ads. Therefore, we will guarantee that brokers will not be able to collect intelligence on patterns of advertising. There will be maintained a ledger of ad service transactions. In it, both the botID and the ad transaction will be stored in encrypted form.

6 Search bot ecosystem and the knowledge coin

We envisage search bots as web representatives of people and organizations. We will provide the means to program search bots for continuous operation, so that, even if the person or organization is inactive, the bot can still search, collect , analyze and produce information, starting from the seeds provided to it by searches from actual humans. In our opinion , search process is very important for production, accumulation, and systematization of knowledge. Search bots will serve vital role in this process. Eventually we envision programming a differential privacy mechanism , similar to the one offered today by Hyperledger Fabric , that would allow bots to exchange

information according to a set of programmable rules, so that arbitrary information transparency rules can be implemented. Bots should be allowed to trade the information they mined from the web. For this we, will implement special currency - knowledge coins. It will come at later stages in the development of the system , when the bot ecosystem is developed enough. We think of search history as a commodity, visibility of which, together with microdetails of information consumption, should be tradable on the exchange. We will provide more details in later versions of this publication.

Search transactions will be recorded in the global time stamped ledger

$$STX =< t,\ q,\ encrypted(botID, time) > \tag{21}$$

If a bot decides to trade some of its search transactions, it reveals keys to the encryption function to his transactions.

7 Algorithms Market

One of the goals of our project is to create open algorithms market, in which teams can sell their ranking, summarization and inference algorithms. This entails flexibility in the use of rank algorithms. For example, there should be a possibility to ask DSs to compute rank according to a new version of an algorithms A (perhaps, a fine-tuned version of the old). For example, developers should have a possibility to deploy and test a new ranking system. For this purpose, we will introduce the Algorithms Market. It is a mechanism that would allow the developers to request computation time from DS of the rank according to their algorithm, and propose these results to brokers in the distributed data center to serve to end users. This service in principle can have many parameters, for example, it can be probabilistic (to test new results). It must be accompanied by means to collect performance data of the new algorithm. We hope that eventually pieces of the system that correspond to the storage, evaluation of rank , and index will become standardized and their relative price will go down, while the importance of algorithms of ranking, summarization and inference will increase. To foster this development it is important to develop technical means to facilitate the access to data from developers. This should be accompanied by a market mechanism where teams can invest in the development of new algorithms.

8 Ledgers and Consensus mechanisms

There are several types of computations that need to be performed by a network of nodes in our system. We propose a fairly simple consensus mechanism: the computation must be performed by several nodes of the same capacity, and if there is agreement on the results , the computation is considered valid. For this to work, it is crucial that the nodes run the same software. We will maintain standard repository of routines (e.g rank calculation according to pluggable algorithm) that nodes can use. This is somewhat analogous to the use of virtual machines in modern blockchain systems.

More importantly, there will be global algorithms that influence the behavior of the system and that sometime need to be adapted to the changing environment. For example, in the index construction , there are many algorithms for the maintanence of hierarchical index which differ in the range of information spread in among the nodes (i.e., the range of visibility of the nodes in the tree). While for performance it is desirable to maintain info on long range connections, it is conceivable that in some situations this information should be hidden (for example, if external entity decides to attack the system , e.g. a government orders its shutdown). For the stability of the system it is vital to have a global mechanism that would allow nodes to change the algorithm that they are using. In the case of indexes, it is not too difficult as we explicitly maintain several overlapping indexes which do not necessarily have the same hierarchical construction. It is more difficult to defend against adversaries that attack the distributed database of DSs with the purpose of physically shutting them down. This is a research we will perform in the future.

9 Planetary Knowledge System

The importance of libraries for human progress hardly needs to be explained. There have been very interesting and positive developments in the development of repositories of scientific data that are widely available. We only mention the recently appeared service by Google where scientific publications from journals are linked to their pdfs. This is accompanied by the availability of excerpts from books delivered by Google. The existence of open preprint servers like arxiv greatly facilitates scientific research. However, the usability of this data is limited. For example, the tools for linking the computations done in scientific papers to more elementary notions are not being developed (with marked exceptions like HOL/Isabelle and LEAN proof assistant systems [**?**, **?**]). One reason for this is the lack of highly structured data stores of web scale. We hope to change this situation by providing open source tools for exploring the semantics of data being stored on the web (including the repositories mentioned

above). This problem involves construction of analogies of abstract syntactic trees for natural language scientific papers.

Most of the technological knowledge is not available on the web. Tech companies guard it. Modern competition system is built in such a way. This way of operation slows down the progress. One of the most notable examples is CPU industry. This situation may be changed by introduction of ways to record and document massive amounts of data, of entire technological systems, and building clear protocols that would allow to trade this data or make it available in situations where there is no danger of competition. We hope that creation of knowledge graph (or rather this dynamical entity from which it is possible to generate adaptive dialogues that could help people navigate complicated landscapes) would facilitate this transition.

10 Conclusion and Timeline

On the practical side, there are many programming issues that have to be addressed. We mention briefly some necessary steps in the software development. The first component is the distributed crawler. Fortunately, this piece of software is very well known and there are in fact many repositories of such crawlers. The second component concerns custom designing distributed data base of pages to be maintained by document servers. Although in principle this is also standard, there are a few custom components - namely, we must implement mechanism to control good local coverage: there must be enough servers in given locality that cover given page. Also, we need distributed database of docserver ids. These structures are not difficult to program.

Most importantly, we need to program hierarchical index construction software, that will optimize the connectivity based on network conditions. Although the index construction by itself is well documented, there are a few novel elements that we need to take care of, in particular , network latency, and the mechanism that will handle node failure. This is a complicated programming task.

Next comes development of distributed ledger systems , front end servers and data balancers, bid auction system, and most importantly virtual machines that will run the chaincode that will be responsible for handling transactions between system components.

We will maintain public repositories of finished software components.

References

[1] Hyperledger Foundation. Hyperledger fabric. `https://hyperledger-fabric.readthedocs.io`, 2019.

[2] EOSIO. Eos white paper. `https://github.com/EOSIO`, 2019.

[3] A.Zomaya Edited by M.Khan, S. Khan. *Big Data-Enabled Internet of Things*. The institute of Engineering and Technology, 2019.

[4] J. Batalla Edited by C. Mavromoustakis, G. Mastorakis. *Internet of Things in 5G Mobile Technologies*. Springer, 2016.

[5] Silvio Micali Jing Chen. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 2019.

[6] J. Bennet. Ipfs- content addressed , versioned p2p file system.

[7] Steem. An incentivized, blockchain-based, public content platform. `steem.com/whitepapper.pdf`, 2018.

[8] et al. E. Ben-Sasson. Zerocash: Decentralized anonymous payments from bitcoin. 2014.

[9] L.Goodman. Tezos - a self amending crypto ledger. 2014.

[10] Protocol Labs. Filecoin: A decentralized storage network. 2017.

[11] Protocol Labs. Libp2p. `https://libp2p.io`.

[12] L.Page S.Brin. The anatomy of a large-scale hypertextual web search engin. *Computer Networks and ISDN Systems*, 1998.

[13] et. al. L.Barroso. The search for the planet: The google cluster. *IEEE Computer society*, 2003.

[14] Tie-Yan Liu. *Learning to Rank for Information Retrieval*. Springer, 2011.

[15] Y.Kim. Convolutional neural networks for sentence classification. *ACL*, 2014.

[16] et al S. Majumdar. Addressing click fraud in content deliverysystems. *IEEE INFOCOM 2007 proceedings*, 2007.