Service Oriented Computing and Applications (2019) 13:95–103
https://doi.org/10.1007/s11761-019-00258-w

ORIGINAL RESEARCH PAPER

Bead Strand Model: a high-efficiency storage structure for self-destructing data in cloud environment

Xiao Fu[1] · Zhijian Wang[2] · Yunfeng Chen[2] · Yihua Zhang[2] · Hao Wu[2]

Received: 7 August 2018 / Revised: 10 December 2018 / Accepted: 20 March 2019 / Published online: 5 April 2019

Abstract
Blockchain prospers in a second mining fever long after the one that occurred in the 1840s. Assumed to be an ark carrying all the data into the brave new world, blockchain provides limited tickets that all the miners struggled to win. Ultimate resources have been wasted to involve the proof-of-work game in order to dig up bitcoins even in countries of the third world. So we strive to design another storage structure with self-destructing mechanism that can provide high-efficiency concurrency and which is more important security for all the users and all types of data. The data would vanish after expiration that could be set by the user in this structure named Bead Strand Model, and this humanitarian mechanism makes it more acceptable than blockchain by the people who concerned privacy and security in cloud environment.

Keywords Bead Strand · High-efficiency storage · Self-destructing · Cloud security

1 Forewords

Since the word blockchain became the buzzword all over Internet from the last three years, it has literally played the role of a leash to the necks of wild economists, stray entrepreneurs, and hungry (maybe also foolish) scientists in the country. The last kind of them, which could have the most perniciousness, is roaming and swarming all the corners, even the remotest ones in this land [1].

In the most undeveloped part of Sichuan province, a medium-size bitcoin mining farm can consume 0.6 billion kilowatt hours electric power each year, the quantity of which is the equivalent of a 100-thousand-population town [2]. More ironically, the State Grid Corporation of China itself is been lobbied by a swarm of so-called blockchain experts to employ blockchain to keep records of such energy contracts to send bills [3].

We have no need to transfer money to some abroad charity foundations via deep web and we do not grow any Cannabaceae plants with some high-power LEDs in our closets, so we have nothing to complain about the miners and the grid. We are terrified by blockchain just like the other techniques that were misused in China, for example, the biggest Chinese AI-driven healthcare system [4] that has mislead a young student patient to death after experimental treatment for synovial sarcoma [5], who knows if the blockchain would reveal how much I paid for my ex-girlfriend's birthday gift to my wife someday in the future (I would rather to die of cancer) because Peoples Bank of China (PBOC) has started to seek a solution to trace every citizens credit records including online payments [6].

What we are afraid of is blockchain would become the shackles around our necks in a not-very-far future.

2 Undeletable problem of blockchain

The storage model of blockchain is a linked list constituted by a set of blocks which are interconnected together with

✉ Xiao Fu
nhri.fuxiao@gmail.com

Zhijian Wang
zhjwang@hhu.edu.cn

Yunfeng Chen
1543391728@qq.com

Yihua Zhang
2801142552@qq.com

Hao Wu
whforhhu@gmail.com

[1] Hohai University and Postdoctoral Work Station of Nanjing Longyuan Micro Electronic Technology Co. Ltd., Nanjing, Jiangsu, People's Republic of China

[2] Hohai University, Nanjing, Jiangsu, People's Republic of China

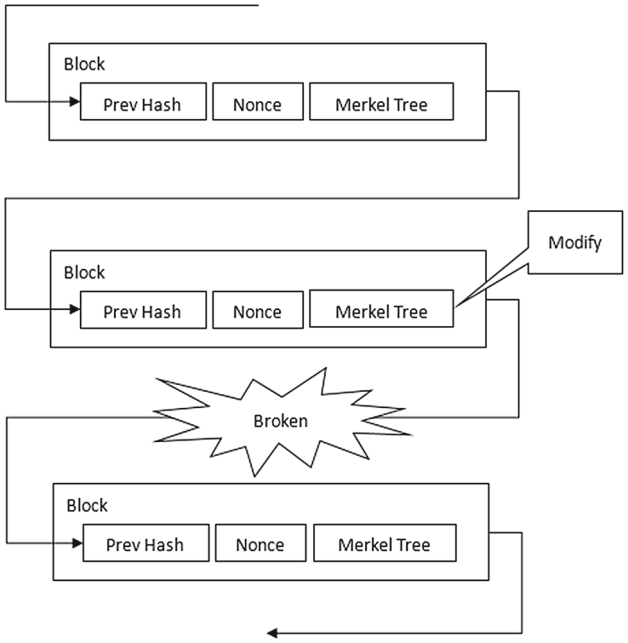

Fig. 1 Broken blockchain caused by modification

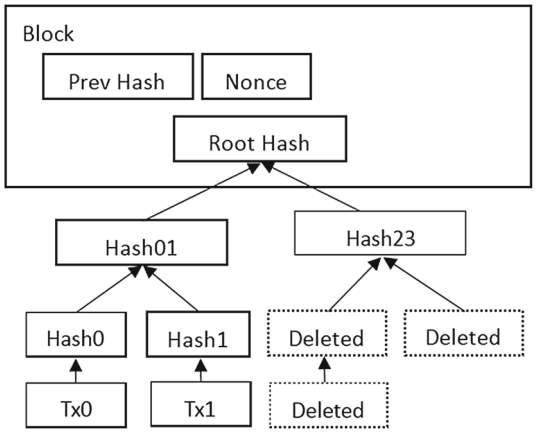

Fig. 2 Delete data from block without breaking the chain

their hash values. Each block is formed by three main parts: a Previous Hash (Prev Hash), a Nonce and a Merkle Tree. The Prev Hash is the hash value of the previous block, in another word an indicator pointed to the previous block. And the Nonce is a nonsense numeric value that was only needed to take part in the hash process; the proof-of-work mechanism of blockchain just uses it to find out which peer could guess the right Nonce faster than the others [7]. That means once the Nonce was found, all the parts of the block would be permanently stored in the peer, broadcasted and accepted by the other ones in the system and could not be modified anymore. If anyone tries to change any bit in the block, its hash value would be changed and the modified block cannot be linked to the next one; the whole chain would be broken into half as shown in Fig. 1.

As we all know, in rational model (RM) the dynamic operation of data contains four types of basic action: select, insert, update and delete. In fact there are only three types: update can be treated as a combination of delete and insert. Blockchain can accept select and insert actions, but delete action is not supported by the storage model of blockchain.

So we can call blockchain an uncompleted (or half-completed) storage model, because we can only append new blocks at the end of list rather than remove any existed block. Archaeologists and auditors may obsess this feature, but as common possibly a little nerdy people we are almost scared to death. The government is planning to keep records of every online purchase (PBOC is a department of State Council, not a real bank) so that our posterity could dig them up and check which brand of baked beans their ancestry bought for breakfast. Though Satoshi Nakamotos manuscript described how to discard out-of-date records to save storage space by using Merkle Tree without changing and breaking the chain as shown in Fig. 2, thanks to Moores Law the cost of storage would not be a problem even in the last several years. Some researchers tried to enhance privacy in blockchain by designing a symmetric encryption scheme which involved distributed hash table (DHT) key-value storage [8]. But no one can guarantee the symmetric encryption algorithm used would never be outdated forever, and the Data Encryption Standard (DES) had already been cracked by a FPGA-based parallel machine about 12 years ago [9].

The reason why current applications do not employ the discarding of out-of-date records was rather ridiculous: Assuming the global mining power remains constant from now on (all the development of computer technologies freezes due to some supernatural power for example), all the blocks would be generated in the year of 2140. Blockchain economists called this a controlled supply because the exhaust of new blocks occurs in a so far future that none of them can witness. But for computer scientists it is a total disaster: That means after 2140 all the storage space would be occupied and no new data could be added in the blockchain. Which is worse, according to Moores Law this day would come much earlier than the one predicted. Nakamotos mechanism was not designed to release the space of blocks, but the Merkel trees. Even after the deletion of Merkel trees, the blocks remain as tombstones of records and could not be recycled and reused. This problem might become another Millennium Bug if not evaluated and approached properly.

In our opinion, some data is better to be buried with our flesh or vanished in shadows of history just like all the unspoken names in this country than to be exhibited to the public on a square. Who does not have one or two skeletons in the closet or Panama Papers?

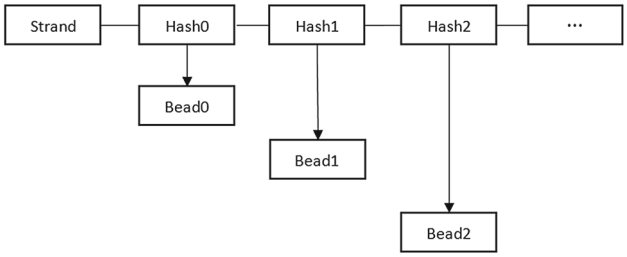

Fig. 3 Bead Strand Model

3 Design of Bead Strand Model

Bead Strand Model, according to its name, divides the data into two parts, Beads and Strand, rather than a single linked list in blockchain. A set of Beads linked together by a specific Strand constituted a byte-array data, for instance, a string, an image or a clip of video as shown in Fig. 3. Unlike the text data stored in blockchain, Bead Strand is a stream-oriented storage structure that not only provides persistence to text, but almost all the types of data ranging from HTML to android package.

To convert a data into Beads and Strand, we designed a four-step procedure which was modified from the original prototype proposed by Fu et al. [10].

First, generate a random key called RandomKey by function Generate (), then encrypt the original input data InputData with RandomKey by using symmetric encryption algorithm Encrypt () and get the encrypted data call CipherData. We chose AES (Advanced Encryption Standard) as the symmetric encryption algorithm to get a balance between security and efficiency.

Second, calculate the hash value CipherHash of CipherData by function Hash (), then split CipherData into pieces of 8 KB length by function Split (); the pieces are called Piece[1], Piece[2] and so on. Combine RandomKey, CipherHash and the byte length of CipherData together as Piece[0] by function Combine (). If the length of any piece is lower than 8 KB, pad it to 8 KB by adding a random byte sequence at the end of it. The reason why we chose 8 KB as the size of piece is, in most Java runtime environment, Node.js for example, the minimal unit of buffer that the memory pool can assign is set to 8 KB in default.

Third, generate a version 4 UUID [11] for each 8 KB-size piece by function GenerateUUID (), tag a timestamp called ExpirationTime on each 8 KB piece. Every Bead is constituted by an UUID, its corresponding piece and ExpirationTime. Use a distributed database to store all the Beads persistently. The database would delete all the expired Beads by comparing ExpirationTime to the current time of network time protocol (NTP) server. Users can set the ExpirationTime of each data they intent to store in Bead Strand Model. If all the Beads of the same data have one specific ExpirationTime, that would cause a collision on attribute: Any attackers who could get access to the database can easily distinguish all the Beads that belonged to the same data from the others. So we use a delete-on-delay mechanism to reduce such a risk: each ExpirationTime would be delayed randomly from 0 to 3600 seconds in order to improve confusion and diffusion of each Bead from all the other ones.

Forth, combine all the UUIDs of Piece[0], Piece[1], Piece[2] and so on in sequence together and get the string called Strand. Any user who has the Strand can get access to all the pieces from the system, decrypt the CipherData by RandomKey and verify the completeness of decrypted data by CipherHash in Piece[0]. But if any Bead has been expired and deleted, even with the right Strand, no one can get access to the Beads and the data stored in them anymore. We add a header of "SDD://" as schema of uniform resource identifier (URI) in front of Strand in order to declare it as a self-destructing data.

Any input data would be expanded by no more than 16 KB: the length of the very first Bead which contained the RandomKey, CipherHash and the byte length of CipherData that is 8 KB. And the last Bead, under the worst-case, only has one single byte with the other 8 KB-1 random byte sequence to pad it. So if the input data were too short, there would be a waste of storage space for the reason that most of the space was taken over by padding data. But notice that this kind of expansion and the size of input data are linearly independent. In stream-oriented scenarios such as images or video clips storage, it is reasonably acceptable.

Compared to DHT, the generation and storage of Beads totally rely on the servers in the application system instead of uncontrolled distributed peers such as Kademlia, BitTorrent and all the other DHT networks. Cloud environment provides higher reliability, availability and performance than P2P system in the level of infrastructure as a service (IaaS): the time cost of discovering resources was always much more than requesting and responding data from peers in P2P system; sometimes it could cost more than several minutes. That is the reason why Bead Strand was born with an advantage in performance compared to DHT technology.

The pseudo-code of the four-step procedure is shown as follows:

```
RandomKey = Generate ();
CipherData = Encrypt (RandomKey, InputData);
CipherHash = Hash (CipherData);
Piece[0] = Combine (RandomKey, CipherHash, Lengthof (CipherData ) );
For n=1 To <Lengthof (CipherData )/(1024*8)>
Piece[n] = Split (CipherData, (n-1)*1024*8, (n)*1024*8);
End For
For Each Piece in Piece[n]
UUID = GenerateUUID ();
```

CreateBead(UUID, Piece, ExpirationTime.addSeconds (RandomDelay.ToSeconds ( )));
Strand = Strand + UUID;
End For
Strand = "SDD://" + Strand;
Return Strand;

When a user converts a data to a strand, all the corresponding Beads of this Strand would have been stored in the distributed database of the system. When the user wants to share this data to any other one, he only needs to send the Strand instead of the whole copy of data. Anyone getting the Strand can request all the Beads of it from the system, decrypt them and validate the decrypted data. If any Bead has been tampered or modified, such as an untrusted peer in distributed database, the hash validate function of the system would identify it and notify the user. If any Bead has met its expiration time and been deleted from the database, the system would notify the user too. So the storage model of Bead Strand is a bit more comprehensive than chainblock: the Beads store cipher pieces and hash values of data, just a little like the Block in chainblock; the Strand contains an array of hash values, indicating relative positions in the cipher text of each corresponding Bead. Unlike in blockchain, Beads are not interconnected, but connected by a Strand externally. No matter whether Beads or Strands are destroyed, the corresponding data could never be revealed to anyone. And all the parts of a Bead, which can be described as triple tuples, are all generated by pseudo-random, or in another word, chaos process:

Bead = UUID, Piece, ExpirationTime.addSeconds (RandomDelay.ToSeconds ( ))

This algorithm can be proofed as a CCA (Chosen Ciphertext Attack) security one if we choose a comprehensive enough hash algorithm, for instant UUID version 4: the symmetric key, hash value and expiration time are all pseudo-randomly and time-correlately generated, even from a same user and a same data as shown in Fig. 4. Therefore, all the Strands generated by the system are linearly independent. So the only possible way to get a data from the system without an available Strand is brute attack, of which the time complexity would be a non-deterministic polynomial (NP) problem. And even with a not-secure enough one, for example MD5 we can still reduce the chance of collision by simply limiting the expiration time to a reasonable period. The shorter we set the expiration time, the lower probability of collision would occur.

Even the UUID version 4 has a slim probability that was almost one in a billion to generate a single duplicate UUID within 103 trillion ones. Unfortunately, the system cannot traverse all the records in order to avoid the collision when

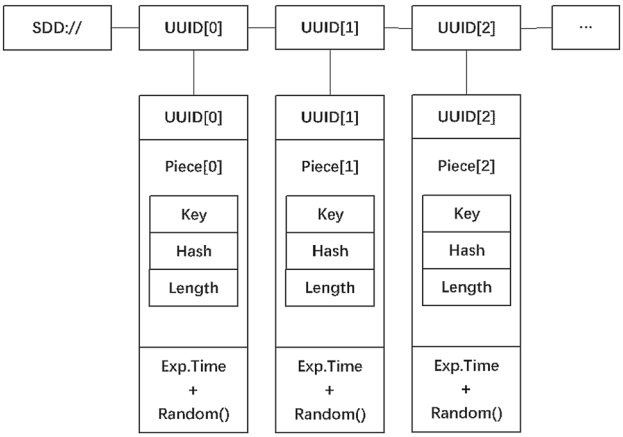

Fig. 4 Data structure of Beads and Strand

each UUID was generated because it would cost much more time and efficiency. So we chose to accept this risk to gain balance between efficiency and security. Maybe in the future, a new version of UUID would solve the problem at the source.

Since we choose 8 KB as the size and version 4 UUID as the identifier of each piece mentioned above, the length of a Strand in ASCII characters could be calculated by the following pseudo-code:

Strand.GetLength ( ) {
Return LengthOf ("SDD://") + 32 * <Lengthof (CipherData)/(1024*8)> + 32;
/* The last 32 characters are the UUID of piece[0] which contains the symmetric key, the hash of ciphertext and length of cipher data. */
}

As we choose AES which is a non-expansion algorithm as the encryption algorithm, the length of CipherData would be equal to the length of InputData. For example, if a user inputs a 500 KB-size data, a very long love letter for instance, into the system, the length of the Strand he would get is : $6+32\times<500/8>+32=2054$. That is not a long text when the user sends it via instant messaging (IM) or email tools to anyone he intends to share this Strand. Blockchain was designed to store small text-based data just like transmission records, so it would be impossible for one single user upload 500 KB data once at a time. But we design Bead Strand Model not only in order to store text-based data, but also pictures, video clips and all the other types of data that people want to share via internet nowadays. So the 2054 would be a real big problem.

In 2014, we have designed a method that exposed a self-destructing data (SDD) link as a uniform resource locator (URL) so that users can embed it into any hypertext markup language (HTML)-based document in the form of a <img>

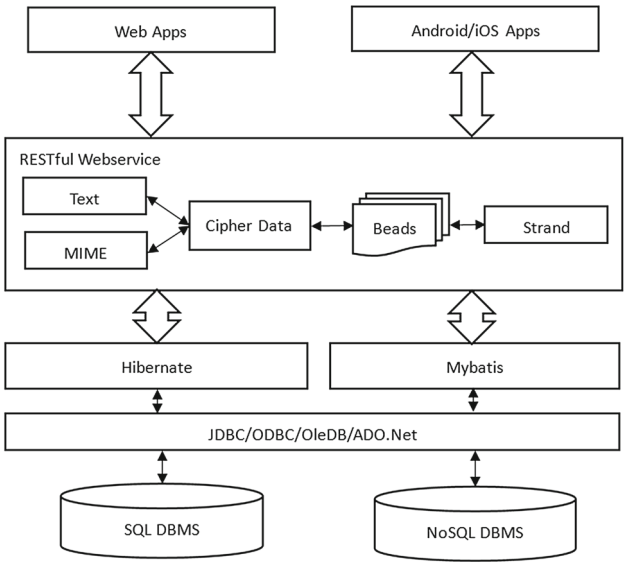

Fig. 5 Structure of prototype system

label [10]. In this method anyone received the SDD link could use any HTML supported web browser to read the content of the SDD data without any other third-party plugins or tools installed. The structure of prototype system is shown in Fig. 5. But as we all know that all the src attributes in HTML labels rely on GET method in hypertext transfer protocol (HTTP) [12]. And the maximum length of request line in GET method is limited to 2048 ASCII characters [13]. That means we cannot expose a Stand with 2054 characters as a HTML label because of its too long length.

There are still several other ways to solve this problem: Geambasu has used a Firefox plugin to convert the cipher data into plaintext [14], and we also developed a plugin for Google Chrome in 2015 to convert a SDD-format URI into decrypted text [15]. But to the users that without those specific browsers installed, the Stands are still unreadable.

So this time we have designed a secondary-index mechanism to help the users who like to embed the Strands into HTML: When a Strand was created, the system would generate an extra single UUID mapping to the Strand. The users can use this UUID as well as the corresponding Strand to get the data. And like the Beads, when the expiration time has come, the system would delete the UUID and the mapping information from the database, too. Each record of mapping information is a triple tuple:

MappingRecord = {UUID, Strand, ContentType }

The attribute of ContentType describes which kind of the data is stored in the Strand. If the user upload a string of UTF-8 characters, the ContentType would be "application/json;charset=UTF-8" in order to declare it as a pure text data just like "text/plain" in MIME (Multipurpose

```
POST / Bead_Strand_Demo/serv/pushContent HTTP/1.1
Host: <HostAddress>:8080
Cache-Control: no-cache
Postman-Token: 5bd763cd-2602-4c53-a292-9ea9b06987c3
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW

------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="deadline"

2018-08-01 12:48:49
------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="content"

hello world
------WebKitFormBoundary7MA4YWxkTrZu0gW--
```

Fig. 6 POST a data of text to the system

```
Content-Type: application/json;charset=UTF-8
Date: Thu, 26 Jul 2018 02:23:00 GMT
Server: Apache-Coyote/1.1
Transfer-Encoding: chunked
{"error":0,"result":"647f71102b7841138ac02acae5eda543"}
```

Fig. 7 Response of request from the system

```
GET /Bead_Strand_Demo/serv/pullContent?cid=647f71102b7841138ac02acae5eda543 HTTP/1.1
Host: <HostAddress>:8080
Cache-Control: no-cache
Postman-Token: 9603442b-ff0d-4f3f-96f3-2ee69f700835
```

Fig. 8 Request of a strand of text from the system

Internet Mail Extensions) types [16]. In Bead Strand Model, we call all this type of data as Content. We provide a HTTP-based servlet interface to deal with the Content uploaded by users. For example, if a user wants to upload a message saying "hello world" which expires at 2018-08-01 12:48:49, the request would be like Fig. 6 if he uses postman to send the POST request.

Once the Beads are stored in the system, a UUID corresponding to the Strand would return from the interface as shown in Fig. 7.

Now we know the UUID corresponding to the Strand is 647f71102b7841138ac02acae5eda543; anyone can get the data of the Strand by using a HTTP GET request. We provide another interface so that users can expose the data as a URI in Fig. 8.

And the system would response the data if none of the Beads has expired yet.

If the user wants to upload pictures, video clips and any other kinds of non-text data, the ContentType would be the same as MIME types [17], for example image/jpeg or video/mpeg4. We call all those types of data as Attachment. Be notice that the Content and Attachment are just dealt by different HTTP interfaces in the system. If a user wants to upload a picture called "pop.png" which expires at "2018-08-25 12:48:49", the POST request would be like in Fig. 9.

```
POST /Bead_Strand_Demo/serv/pushAttachment HTTP/1.1
Host: <HostAddress>:8080
Cache-Control: no-cache
Postman-Token: a1d52b3e-07cb-46ec-8402-2626fe598281
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW

------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="deadline"

2018-08-25 12:48:49
------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="file"; filename="pop.png"
Content-Type: image/png


------WebKitFormBoundary7MA4YWxkTrZu0gW--
```

Fig. 9 Post a data of picture to the system

```
GET
/Bead_Strand_Demo/serv/pullAttachmentStream?aid=85450d2b54fb48d39bc09f1347a057c0
HTTP/1.1
Host: <HostAddress>:8080
Cache-Control: no-cache
Postman-Token: 325a8014-f245-4a7c-af91-4c1f60299607
<This is the binary stream of the picture we requested.>
```

Fig. 10 Response of a strand of picture from the system

The response containing an UUID which is 85450d2b-54fb48d3-9bc09f13-47a057c0 would be received. So we can use the UUID to request the picture before its expiration by sending GET request to the URI in Fig. 10.

Notice that for employing secondary-index UUIDs instead of the Strands, the length of each secondary index is always fixed 32 ASCII characters and is far low from 2048. Now users can easily upload and request almost all the kinds of data they are interested in without worrying about the out-of-size problem. This sounds much funnier and inspiring than the text-only blockchain.

Secondary-index UUIDs would introduce risk in exposing Strand to any other users such as attackers if the database which MappingRecords store in has been infiltrated. Currently, we have not designed any specific authentication mechanism for Bead Strand Model. The challenge response of this model is rather simple: any user who has the available Strands could get full access to the plaintext. So if an attacker gathers some secondary indexes, the corresponding plaintexts would be exploited. We strongly suggest to keep them securely in private or trusted servers protected by server-level authentication or domain authentication mechanism, instead of infrastructure systems.

4 Implement and analysis

We have implemented the prototype system of Bead Strand Model on J2EE platform. Though the former prototype works fine with MySQL database according to Wus experiment [15], we chose to run the test on a real distributed storage environment this time. So we built up a Real Application Cluster (RAC) with 3 nodes to be the data access layer of system. Each node had the same hardware and software configuration: Inter Core Duo2 CPU, 8 GB RAM, 6TB HDD, 1 GB Ethernet Interface, Ubuntu 16.04 OS. And we deployed the business logic layer on another node with Tomcat, and the configuration was: Inter Core Duo2 CPU, 8 GB RAM, 256G SSD, 1 GB Ethernet Interface, Microsoft Windows Server 2003 R2 Enterprise, Tomcat 8 Web Server.

The efficiency experiment process was simple: randomly post 10 billion pieces of data into the system and then randomly request 10 thousand ones from the system.

We chose Oracle 11g and Apache Cassandra 2.0.9 as the database management system (DBMSs), and the size-related average latencies of posting and requesting are shown in Figs. 11 and 12.

In posting latency, Cassandra showed 50% lower time cost than Oracle as well as in requesting. Notice that we set the replication factor of Cassandra to 1 in fairness. Theoretically, if we increase this number, the performance would be improved further. Both SQL and NoSQL DBMS showed a good efficiency in the prototype system of Bead Strand Model, with per users thread in SQL DBMS. And in NoSQL this capability would be doubled.

With increasing the number of hashes, the efficiency of generating new blocks would be gradually decreased in blockchain because the difficulty of proof-of-work challenge would exponentially grow. But in Bead Strand Model the hash of Beads is randomly and time-correlately generated by the system, and distributed according to user's needs. Since the quantity of UUID is so large that no one wants to possess them in private, there is no need to play the stupid proof-of-work game to wasted power and the other valuable resources. And when a Bead expires, the UUID might be recycled and reused by the system for the other users; there is no crisis of exhausted hash in foreseeable future.

Because of the strong consistency of Oracle, its performance in high concurrency was far lower than NoSQL BDMS. So we used MongoDB and Cassandra to do the test of concurrency.

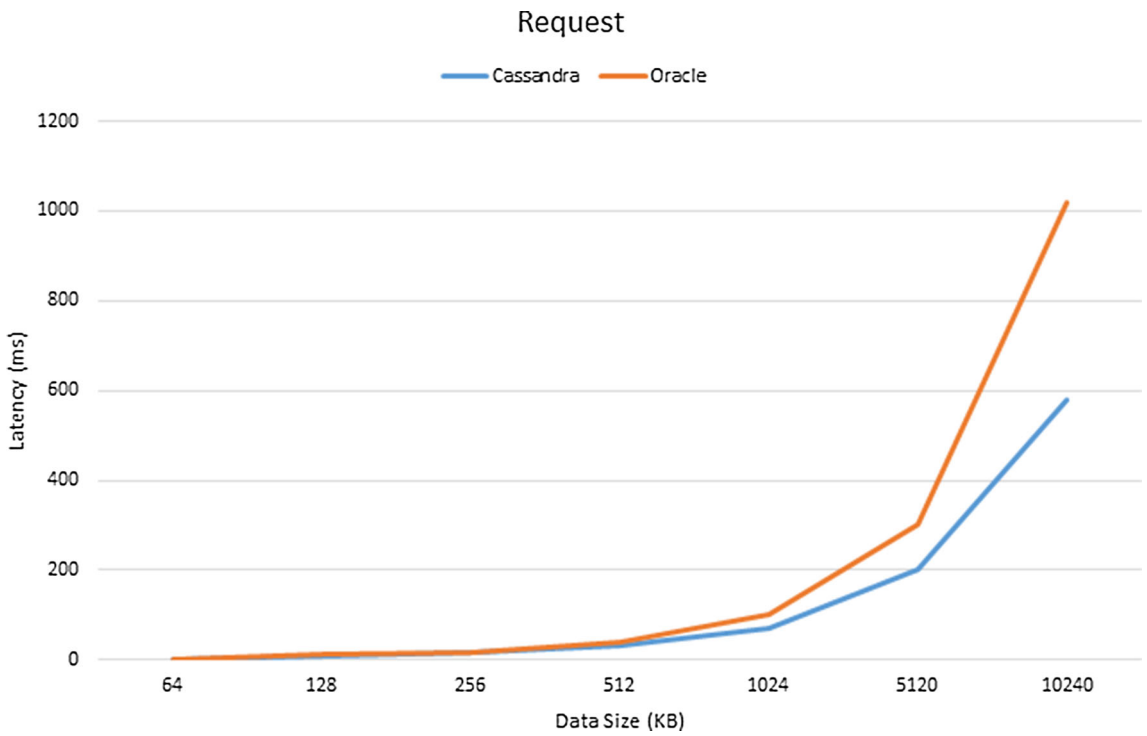

Fig. 11 Request latency of the system

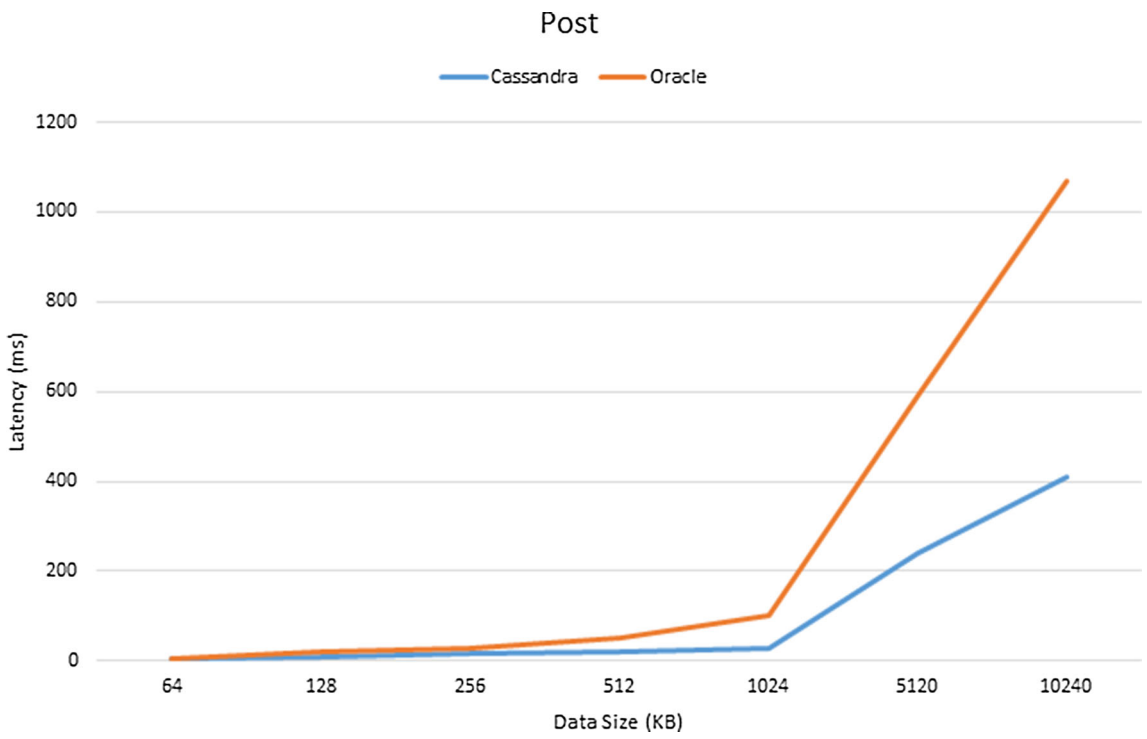

Fig. 12 Post latency of the system

In a concurrency of 100 threads, the average 2000 operations (95% request and 5% post) latency of MongoDB and Cassandra was 40 and 10 milliseconds, respectively, as shown in Fig. 13. That means the maximum number of concurrent users the prototype system can support could be more than 100 nearly in real time.

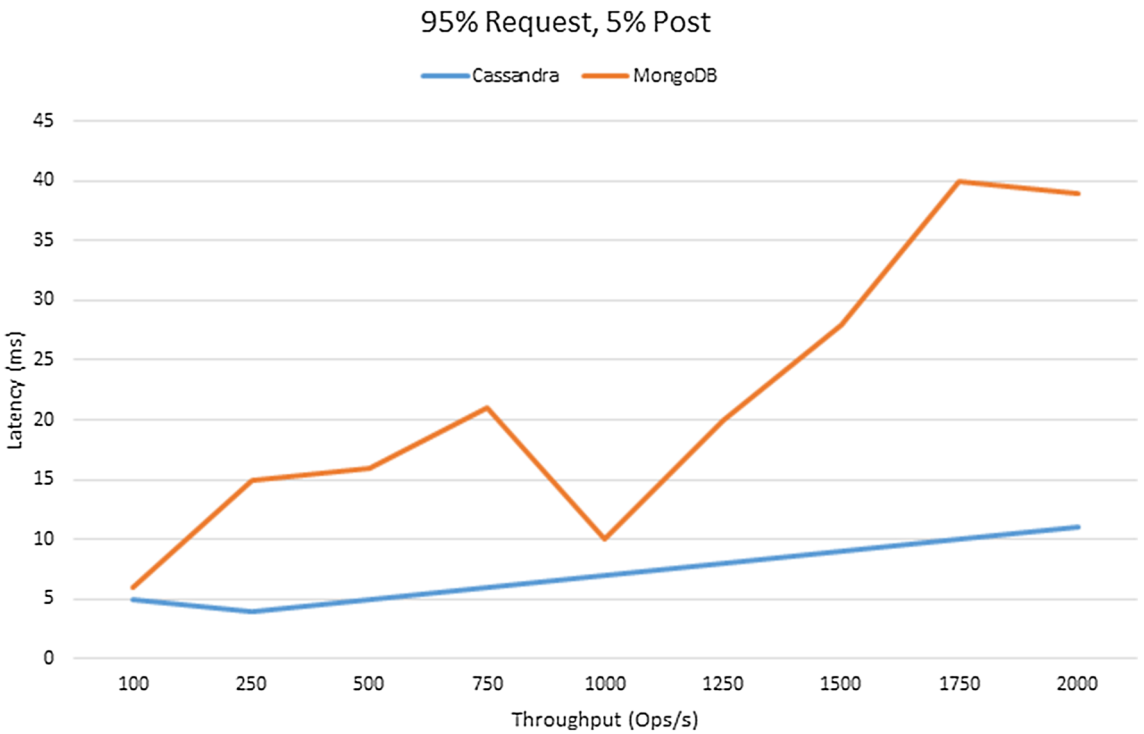

Fig. 13 Throughput latency of the system

As shown in the experiment results, Bead Strand Model has both good efficiency and concurrency in performance, and what is more important, could provide delete operation for the data while blockchain failed to.

5 Conclusion

Obviously, the data lives much longer than mankind. Thousands of years ago, people who seek eternity started to keep their records on all the kinds of media ranging from clay tablets, granites, golden leaves, to blockchains in the big data era. Too much resources have been squandered to calculate less and less hashes which were considered to be the ticket for becoming immortal via internet. So we strive to give a solution for the other people not interested in the blockchain fever, a storage structure for self-destructing data which would only store their data in a limited time.

Anyone can upload their files, images and video clips without playing stupid proof-of-work games thanks to the ultimate and recyclable UUIDs. And this mechanism is proofed Chosen Ciphertext Attack security by using time-related random encryption algorithm. Both for SQL and NoSQL DBMSs, this structure provides high efficiency at a reasonable cost which makes them practical for geek individuals and enterprises, especially in cloud environment.

Stream-oriented storage is the best application scenario as we imaged: media streams such as video and audio require more usability and efficiency than transaction records. And unlike the transactions, media streams embed in SNS messages always have no needs for permanent storage, Snapchat for example. These features make them the perfect potential applications for Bead Strand Model.

Acknowledgements This paper was supported by the Jiangsu Provincial Water Conservancy Science and Technology Project (2018056), the Fundamental Research Funds for the Central Universities (2016B14014), the Six Talent Peaks Project in Jiangsu Province (RJFW-032) and the Priority Academic Program Development of Jiangsu Higher Education Institutions (PAPD).

References

1. Peck ME (2017) Why the biggest bitcoin mines are in China. In: IEEE spectrum
2. Wang C (2017) A visit to a bitcoin mining farm in Sichuan. China Reveals Troubles Beyond Regulation. www.bitcoin.com
3. Zhang N, Wang Y, Kang C (2016) Blockchain technique in the energy internet: preliminary research framework and typical applications. Proc Chin Soc Electr Eng 15(36):4011–4022. https://doi.org/10.13334/j.0258-8013.pcsee.161311
4. Liu Q, Chen T, Cai J (2012) Enlister: baidu's recommender system for the biggest Chinese Q and A website. In: ACM conference on recommender systems. ACM, New York, pp 285–288. https://doi.org/10.1145/2365952.2366016
5. Abkowitz A, Chin J (2016) China launches Baidu probe after the death of a student. Wall Street J 20160502. https://www.wsj.com/articles/china-launches-baidu-probe-after-the-death-of-a-student-1462209685

6. Liu C (2017) Research on the application of block chain technology in the construction of social credit system in China. Credit Ref 8:28–32
7. Nakamoto S (2008) Bitcoin: a peer-to-peer electronic cash system. https://bitcoin.org
8. Zyskind G, Nathan O, Pentland AS (2015) Decentralizing privacy: using blockchain to protect personal data. In: IEEE security and privacy workshops. IEEE Computer Society, New York, pp 180–184. https://doi.org/10.1109/SPW.2015.27
9. Kumar S, Paar C, Pelzl J (2006) How to break DES for Euro 8,980. In: 2nd workshop on special-purpose hardware for attacking cryptographic systems, Cologne
10. Fu X, Wang ZJ, Wu H (2014) How to send a self-destructing email: a method of self-destructing email system. In: IEEE international congress on big data. IEEE Computer Society, New York, pp 304–309. https://doi.org/10.1109/BigData.Congress.2014.51
11. Leach PJ, Mealling M, Salz R (2005) RFC 4122: a universally unique identifier (UUID) URN namespace. IETF. https://doi.org/10.17487/RFC4122
12. Berners-Lee T, Connolly D (1995) RFC 1866 hypertext markup language-2.0. IETF. https://doi.org/10.17487/RFC1866
13. Fielding R, Gettys J, Mogul J (1999) RFC 2616: hypertext transfer protocol—HTTP/1.1. IETF. https://doi.org/10.17487/RFC2616
14. Geambasu R, Kohno T, Levy AA (2009) Vanish: increasing data privacy with self-destructing data. In: 18th USENIX security symposium, pp 299–316
15. Wu H, Fu X, Wang ZJ (2015) Self-destructing data method based on privacy cloud. In: International conference on logistics engineering, management and computer science (LEMCS 2015), pp 1207–1211. https://doi.org/10.2991/lemcs-15.2015.241
16. Freed N, Borenstein N (1996) RFC 2045: multipurpose internet mail extensions (MIME) part one: format of internet message bodies. IETF. https://doi.org/10.17487/RFC2045
17. Freed N, Borenstein N (1996) RFC 2046: multipurpose internet mail extensions (MIME) part two: media types. IETF. https://doi.org/10.17487/RFC2046