A Blockchain Approach for Data Transparency in a Relational Database System

by

Mohammadamin Beirami

A thesis submitted in partial fulfillment
of the requirements for the degree of

Masters of Science

in

Computer Science

University of Ontario Institute of Technology

Supervisors: Dr. Ken Q. Pu and Dr. Ying Zhu

August 2018

Abstract

This thesis deals with the development of a framework based on Blockchain technology that is implemented on top of a Relational Database Management System and makes it extremely difficult for the privileged or unprivileged users of the system to conceal their activities on a relational database. We present a mechanism to audit the activities of the users, verify the validness of submitted transactions on the database and create a Blockchain out of submitted transactions. We also present a practical solution to handle large query workloads on the temporal audit tables. By implementing this framework, not only proof of work is available, but also malicious activities such as intentional or unintentional fake data manipulation could be discovered and reported to the moderator of the system.

Acknowledgements

Contents

List of Figures

List of Tables

Listings

Chapter 1

Motivation and Problem Definition

1.1 Motivation

Today, data is seen as the lifeblood of organizations that is helping them to make strategic decisions more efficient and perform the operations faster. Organizations store their highly confidential data, such as financial documents, customer information, medical records and more in the form of data records in a database and later, use them for their sensitive operations. However, in our current connected era, cyber-security is one of the biggest challenges of organizations. Databases, could be accessible on the public networks where adversaries utilize hacking techniques to manipulate the data that are stored in them. Offline databases also are not safe as the attacks might be carried out within the organization by privileged users of the system. The fake data that are the result of such malicious activities, if not identified on the database, may result in irreversible consequences. As a result, organizations and businesses spend a considerable amount of money to utilize cyber-security techniques in order to make their stored data safe and reliable.

Traditionally, malicious activities on a system including fake data manipulation to the database is prevented by restricting the activities of the users on the system. Also cryptographic techniques such as data encryption or electronically signing the data has proven effective in identifying fake data manipulation. This is primarily done by assigning a pair of cryptographic keys to the users, by which they can securely encrypt or decrypt the data and submit it to the database. Fabricating cryptographic keys is computationally infeasible, hence it is extremely difficult for an outsiders to manipulate data inside the databases without notice. The downside of these techniques is that it requires to fully trust the activities of authenticated users which in turn, brings up a lot of security concerns. Also with ever-increasing complexity of cyber-criminal techniques, each day a new approach to penetrate the database systems is identified. Hence, it is naive to assume that access restriction or cryptographic techniques alone could solve the issue.

Therefore a system is needed to confirm the reliability of data based on verifiable evidences and not by relying on trust. This requires that the transactions on the database system to be transparent, meaning that for a record in a database throughout its life-cycle, it should be evident that its data has always been generated and modified by official sources. To achieve this, the system not only should be able to identify and report the data generated from unrecognized sources but also it should be able to show the proof of work done by privileged users. By providing proof of work, all users who interfered with data manipulation in a database system are identified and their activity information is reported.

In this thesis we have developed a system which ensures transparency of activities in a relational database system. Our developed system identifies and reports any malicious data manipulation by outsiders and provides proof of work by referring to the history of transactions for any records stored in the database. History of records are temper-proof and is protected by cryptographic techniques. We also developed a mechanism to handle large query workloads on the historical records. There were four main challenges that needed to be addressed while developing the system: Auditing the transactions on the database system, handling large query workloads on the audit tables, verifying the validness of query results and making the audit tables to be temper-proof.

1.2 Problem Definition

Given a relational database D, let r to be the relational table in D. Denote attributes of r as $attr(r)$. Assume $attr(r) = [id, m, u, sig(m|u)]$ where id is the id of records in the database, $m = [col_1, ..., col_n]$ is n number of data columns in r, u is the information of the user who submitted the record and $sig(m|u)$ is the digital signature of the record submitted by u. Also let D^T be the temporal database in which the history of the records in D is stored. we denote r^T as the tables in D^T. Assume that the attributes of r^T are $attr(r^T) = [id, m, u, sig(m|u), t, d]$ where t and d are the timestamps in which the record was created/updated or deleted respectively.

A submitted transaction is said to violate transparency, hence untrustworthy in any of the following scenarios:

Scenario 1. Let q be the result of the query $q = \sigma_{(id)}(r)$, which is the record submitted by the user u. The result of query is untrustworthy if $\{q[sig(m|u)] : sig(m|u) \in r\} \neq sig(\{q[m] : m \in r\}|u)$. That is, by digitally signing the record m with the u's cryptographic keys, we get a different result than the submitted signature to the table. The reason that this scenario may happen is that:

- The record was altered accidentally or maliciously.
- A user maliciously claims to be one of the privileged users of the system with

fake credentials.

Scenario 2. Let q be the result of query $q = \sigma_{(id)}(r)$. The result of query is untrustworthy if $q[u \vee sig(m|u)] = \emptyset$. In other words, the resulted record was submitted by an anonymous user to the database.

Scenario 3. Given a particular timestamp t_0, the result of query on the temporal database $q^T = \sigma_{(id,t=t_0)}(r^T)$ is untrustworthy if $\{q^T[sig(m|u)] : sig(m|u) \in r^T\} \neq \{sig(q^T[m] : m \in r^T\}|u)$ and if $q^T[u \vee sig(m|u)] = \emptyset$. This means that a former transaction on the record that occurred in t_0, was submitted illegally.

Scenario 4. Given the current timestamp t_{max}, let $q^T = \sigma_{(id,max(m|t_{max}):m\in r^T)}(r^T)$ to be the latest version of a record in D^T and $q = \sigma_{id}(r)$ to be the same record in D. A record is said to be untrustworthy if $q^T \neq q$ meaning that the latest version of a record in the temporal table does not match the record in a normal table. This include the following cases:

- $q^T[m] \neq q[m]$
- $q^T[sig(m|u)] \neq q[sig(m|u)]$
- $q^T[d|id] \neq \emptyset$ but $q[id] \in r$
- $q^T[d|id] = \emptyset$ but $q[id] \notin r$

All in all, we argue that the data in a database is said to be transparent if:

1. The content of the records match the submitters' digital signature.
2. No anonymous transaction was submitted to the system.
3. History of applied transactions is provided for all records.
4. Items 1 and 2 are valid for all former transactions on the records.

5. The latest version of the records in the temporal audit table match the records in the normal table.

1.3 Contributions of the Thesis

1.4 Outline of the Thesis

Chapter 2

Background Knowledge

This chapter introduces the tools and concepts that the proposed system is based on. In the first section the concepts and tools that were utilized from the Relational Database. Second section covers the basics of cryptography and hashing techniques and finally in the third section we introduce the basics of Blockchain technology.

2.1 Relational Database

Definition 1. (Temporal Database): Let $r_i = r_1, r_2, ..., r_n$, be n number of tables in the relational database D. Denote the attributes of each table as $attr(r_i)$ where $r_i \in D$. A temporal table denoted r_i^T is a table with attributes $atr(r_i^T) = \{(updated, deleted) \in \mathcal{T}\} \cup attr(r_i)$ where $\mathcal{T} = t_0, t_1, ..., t_n$ is the time domain in which transactions on r_i happened. A temporal database denoted D^T is the result of augmenting D by r_i^T:

$$D^T = D \cup \{r_i^T : r_i \in D\}$$

The temporal database D^T contains the entire history of the records ever existed in D.

Table 2.1: Normal Relational Table r_1

id	item	value
22	Pencil	7.50$
23	Notebook	12.0$

Table 2.2: Temporal Table r_1^T

id	item	value	updated	deleted
21	Ruler	3.25$	2018-02-10	-
21	Ruler	3.25$	-	2018-02-20
22	Pencil	8.0$	2018-03-21	-
22	Pencil	7.50$	2018-03-30	-
23	Notebook	12.0$	2018-04-01	-

Example 1. Given the normal relational table r_1 (Table 2.1) and temporal table r_1^T (Table 2.2), the $attr(r_1) = (id, item, value)$ and $attr(r_1^T) = (id, item, value, updated, deleted)$. The result of query $q_1 = \sigma_{id=22}(r_1)$ is $\{(22, Pencil, 7.50\$)\}$ and the result of same query on the temporal table $q_2 = \sigma_{id=22}(r_1^T)$ is $\{(22, pencil, 8.0\$, 2018{-}03{-}21, NULL), (22, pencil, 7.50\$, 20$ $03-30, NULL)\}$. Also the query $q_3 = \sigma_{id=21}(r_1)$ results in $NULL$ however, the same query on the temporal table $q_4 = \sigma_{id=21}(r_1^T)$ has the history of record with id $= 21$: $\{(21, ruler, 3.25, 2018-02-10, NULL), (21, ruler, 3.25, NULL, 2018-02-20)\}$.

Definition. (Auditing Activities):

Definition2. (Time domain): The time domain $\mathcal{T}$ consists of discrete timestamps $t_0, t_1, ..., t_n$ in which transactions on tables $r_i \in D$ happened. the range of time domain is: $\mathcal{T} = [t_0, \infty)$ where t_0 is the timestamp in which the first record added to the table r_i. The time domain of a temporal table $r_i^T \in D^T$ is calculated by:

$$\mathcal{T} = r_i^T[updated] \cup r_i^T[deleted]$$

For example in the temporal table r_1^T (Table 2.2), the time domain is: $\mathcal{T} = [2018-02-10, 2018-04-01]$.

Definition 3. (Timestamps): A timestamp $t_i \in \mathcal{T}$ is a particular position in the time domain, in which particular transaction(s) happened. For example in the temporal table r_1^T (Table 2.2), "2018-03-30" is a timestamp in which the record with "id = 22" updated.

Definition 4. (Timeline): Let $u_i(t_j)$ be the total transactions on the tables $r_i \in D$ at timestamps $t_j \in \mathcal{T}$, where $i, j = \{0, 1, ..., n\}$. The $u_i(t_j)$ could be represented as an ordered set points on a vector. This vector is called a timeline for the transactions happened on $r_i \in D$. Figure x illustrates the concept of timeline.

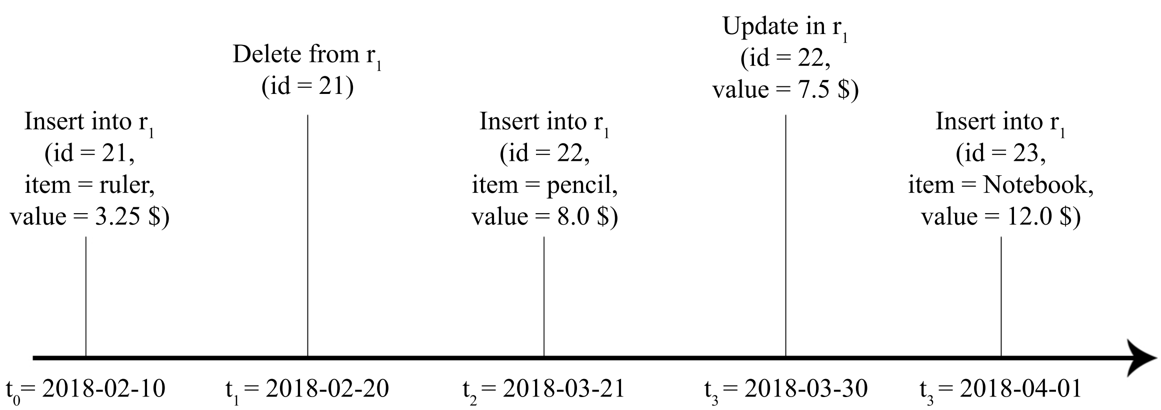

Figure 2.1: Timeline.

Defionition 5. (Snapshot and snapshot-queries): Given a temporal table $r^T \in D^T$ and a timestamp $t \in \mathcal{T}$, we denote $s(t)$ to be the table instance that obtained by calculating the $\{max(r^T[m])|t : m \in r)\} - r^T[deleted]$ for $\mathcal{T} \leq t$. Note that $s(t) \in D^T$ but not necessarily $s(t) \in D$. A snapshot is a materialized version of

$D(t) = \{s_1(t), s_2(t), ..., s_n(t)\}$. A snapshot-query is an arbitrary relational query on $D(t)$.

We can construct the snapshots using simple windowing functions (as in supported by PostgreSQL [**?**]).

snapshot(r, t) =
WITH T AS (
SELECT id, {last_value(x) as $x : x \in attr(r)$} OVER W
FROM r^T
WHERE updates $\leq t$
WINDOW W AS PARTITION BY id ORDER BY updates
)
SELECT id, $\{x : x \in attr(r)\}$ FROM T
WHERE NOT T.deleted

The query snapshot(r, t) computes the snapshot of r at timestamp t by applying the latest update of each tuple up to timestamp t, while removing tuples that have been deleted.

Proposition 1. Linear Time: Assume that the tables are updated at a constant rate over time, then the complexity of snapshot(r, t) is

$$\mathcal{O}(|\{x : x \in r^T \text{ and } x.\text{updates} \leq t\}|) \simeq \mathcal{O}(t)$$

Note that this complexity is also valid for regular query answering (id, t) where the latest version of a particular record at time t is requested.

2.2 Cryptography

Cryptography is a way of secure communication between parties in a network while adversaries might also be present. Using Cryptography, messages sent or received are encrypted so that the adversaries cannot read the normal form of the message. This communication is established through various steps such as cryptographic key assignment, encryption and decryption of messages or digitally signing a message and verifying the digital signatures.

Definition.(Cryptographic Keys): Let u be the authenticated user who is using database D. By the creation of the profile of u in the system, a set of strings $< K_{priv}, K_{pub} > \in \mathbf{N}^{+}$ is generated and assigned to the user where K_{pub} is the public key that is accessible to everyone on the system, and K_{priv} is the private key that is known only to u. These keys are used to encrypt/decrypt messages which is transmitted between the users.

Definition.(Assymetric Encryption): Let E be the encryption algorithm, D be the decryption algorithm, m be the message which needs to be encrypted and c be the encrypted message. Given the cryptographic keys $< K_{pub}, K_{priv} >$, An encryption technique is said to be asymmetric if:

$$c = E(K_{pub}, m) \text{ and } m = D(K_{priv}, c)$$

or

$$c = E(K_{priv}, m) \text{ and } m = D(K_{pub}, c)$$

Therefore:

$$D(E(m, K_{pub}), K_{priv}) = m$$

and

$$D(E(m, K_{priv}), K_{pub}) = m$$

Note that, if K_{pub} is known, and $E(K_{pub}, m)$ is also known, in asymmetric encryption method, it is impossible to get m without K_{priv}.

Figure x shows the basic steps to send and receive messages between two parties in a secure way by utilizing asymmetric encryption technique. [Figure and description to be added]

Definition. (Hash function): Assume $m \in D$ to be the message with an arbitrary size chosen from domain D. $hash(m) \rightarrow sketch$ is a hash function that maps the m of any size to a fixed size string (normally 256 bits).

Definition. (Digital Signature): Digital signature is used to ensure that the digitally transfered data has not altered while transferring. Also it verifies whether or not the transfered data was submitted by a recognized source.

Let m be the document which needs to be digitally signed and transfered. In order to digitally signing a document and verify a signature, following steps should be taken:

- **Step 1.** $S_r = h(m)$
- **Step 2.** $c = E(S, K_{priv})$
- **Step 3.** m and c are sent

In order to verify:

- **Step 1.** m and c are received
- **Step 2.** $S_t = h(m)$ is calculated
- **Step 3.** $S_r = D(c, K_{priv})$ is obtained
- **Step 4.** m is valid if $S_t = S_r$ and invalid if $S_t \neq S_r$

Figure x depicts the steps that needs to be taken for digitally sign a document and verifying a digital signature.

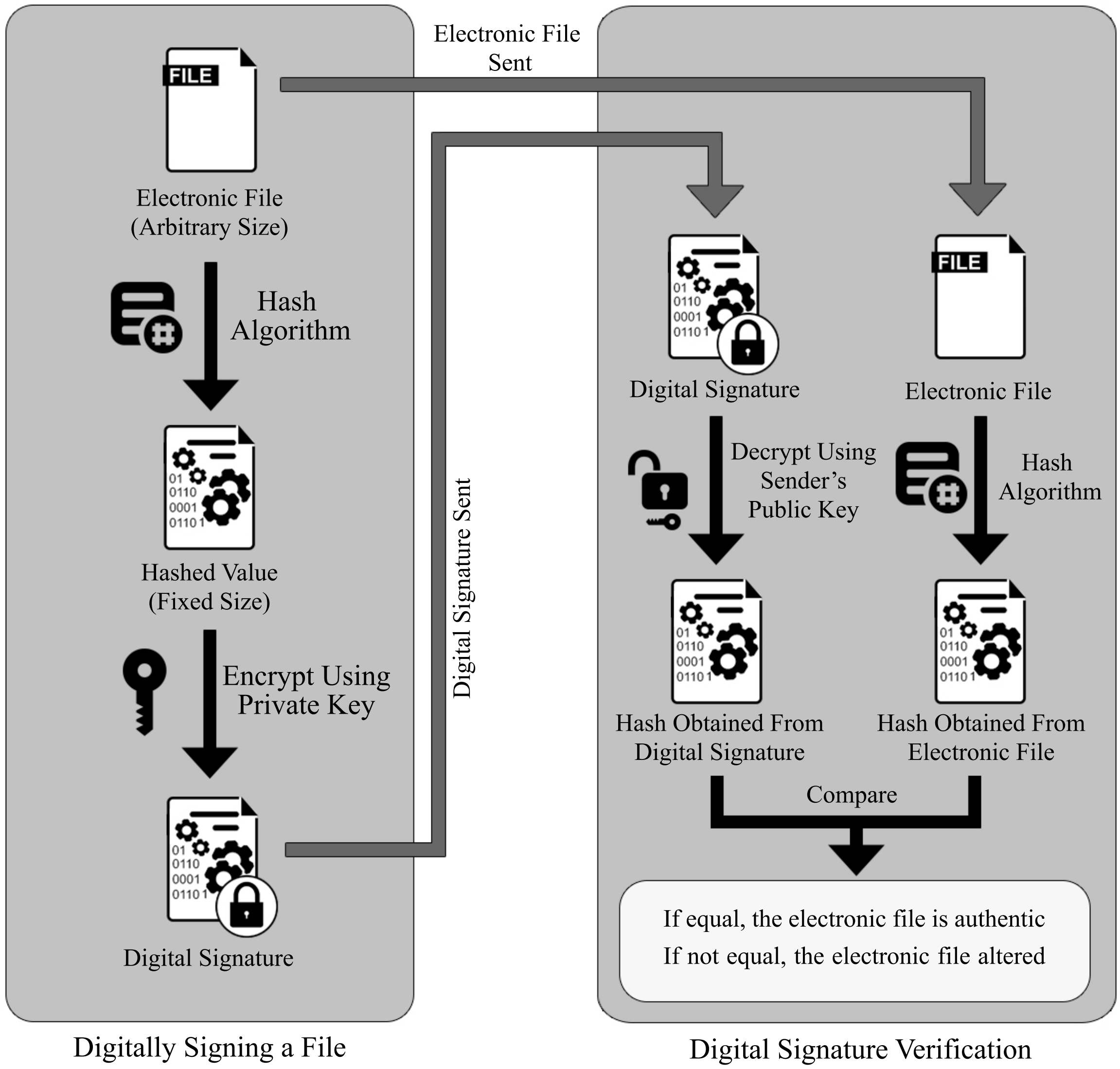

Figure 2.2: Digital Signature Creation and Verification.

2.3 Blockchain

hash pointers

Components of a Block

Chapter 3

Approach

In this section we extensively talk about the algorithms and methods that we have used in our project.

3.1 Overview

To be written... in this section the whole procedure should be described, understandable to everyone! the whole system should be sketched... and each section of the system should be described.

3.2 Creating Temporal database

3.2.1 Auditing

3.2.2 Large query handling On the Audit Tables

As discussed in proposition 1, reconstructing a snapshot table and computing the latest version of a particular record at a timestamp of interest by using a temporal audit table is a linear time process with time complexity of approximately $\mathcal{O}(t)$. Therefore, in the presence of multiple and concurrent queries on the temporal database, such transactions are computationally expensive and inefficient. Running queries on the temporal audit database and reconstruct historical tables to authenticate the proof of work of the records is a routine process in the proposed system. Hence a practical solution is required to handle large query workloads on the append-only temporal audit tables.

With reference to [cite][cite], using precomputed materialized view is proven to be effective in large query handling. Therefore, precomputing m number of snapshots for materialization, seems to be helpful in lowering the cost of queries on the temporal audit tables in the proposed system. However, the question which needs to be answered is to find the optimal timestamp on the timeline of the temporal database, where all the queries could benefit from materialization of snapshots.

Definition. (Query Answering Using Snapshots): Let q_0 be a single query on a temporal database D^T. It is argued that precomputing $\mathit{snapshot}(r, t)$ using the method described in Definition 5, and use it as the materialized view to answer q_0, can decrease the computational cost.

Proposition: Suppose there is a materialized precomputed $\mathit{snapshot}(r, s)$ available on the timeline, then queries and subsequent $\mathit{snapshot}(r, t)$ could be calculated with complexity:

$$\mathcal{O}(|\{x : x \in r^T \text{ and } x.\text{updates} \in [s, t]\}|) \simeq \mathcal{O}(|s - t|)$$

Proposition: Let $T_q = \{q_1, q_2, ..., q_n : q_i \in \mathcal{T}\}$ to be n number of old queries which based on them the optimal position of snapshots were calculated. Denote $T_q^n = \{q_1^n, q_2^n, ...q_c^n : q_i^n \in \mathcal{T}\}$ as c number of new queries performed on the timeline of the temporal database and k be the maximum allowed number of new queries that can use *snapshot*(r, s). When $|T_q^n| > k$, the system recalculates the optimal timestamps for the snapshots based on $T_q \cup T_q^n$ and dynamically adjust the location of the snapshots on the timeline.

3.2.3 Materialization of Snapshots

Let $T_q = \{q_1, q_2, \ldots, q_n : q_i \in \mathcal{T}\}$ be the timestamps of n queries, each querying the database at $D^T(q_i)$. Our goal is to compute m number of snapshots in optimal timestamps on the timeline to answer to T_q at lowest possible cost. The cost function is defined as the total query answering cost given m number of precomputed snapshots.

Definition. (Cost of Query Answering): In the presence of a single materialized precomputed snapshot at timestamp $s \in \mathcal{T}$, the cost of answering the query T_q is calculated as:

$$\mathrm{cost}(T_q|s) = \sum_{q \in T_q} |q - s|$$

Now if multiple snapshots at timestamps $S = \{s_1, s_2, \ldots, s_m : s_i \in \mathcal{T}\}$ were precomputed and materialized, then

$$\mathrm{cost}(T_q|S) = \sum_{q \in T_q} \min\{|q - s| : s \in S\}$$

Definition. (Optimal Snapshot placement): The goal is to precompute m number of snapshots at optimal timestamps $S = \{s_1, s_2, \ldots, s_m : s_i \in \mathcal{T}\}$ for materialization, such that

$$cost(T_q|S) = Argmin(\sum_{q \in T_q}\{|q - s| : s \in S\})$$

Optimal Materialization of a Single Snapshot

While the goal is to precompute multiple number of snapshots at optimal timestamps for materialization, at the first stage, finding an optimal timestamp for a single snapshot s^* is discussed and in the subsequent sections, several approaches to find optimal timestamps for multiple number of snapshots will be proposed and compared.

Proposition: The optimal position for a single snapshot on the timeline for materialization is $s^*(T_q) = median(T_q)$ which can be computed in $\mathcal{O}(|T_q|)$.

Proof of the proposition: At first, the placement of a single snapshot for two queries are discussed and then the argument is generalized for multiple queries: Assume that there are two queries $T_q = \{q_1, q_2 \ : q_i \in \mathcal{T}\}$ on the timeline, such that, $q_1 < q_2$. for the placement of a single snapshot $s^* \in \mathcal{T}$ on the timeline, there are several cases which needs to be considered:

Case 1: $s^* \in [q_1, q_2]$, hence $q_1 \leq s^* \leq q_2$. in this case, the cost is:

$$cost(T_q|s^*) = \sum_{i=1}^{2} |q_i - s^*| = (s^* - q_1 + q_2 - s^*) = (q_2 - q_1)$$

from above, it is concluded that the cost of running two queries q_1 and q_2 when snapshot is placed between them, is equal to the deviation between the two queries.

case 2: $s^* \notin [q_1, q_2]$ and $s^* < q_1 < q_2$. for this case the cost could be calculated as follows:

$$cost_T(T_q|s^*) = \sum_{i=1}^{2} |q_i - s^*| = (q_1 - s^* + q_2 - s^*) = (q_1 + q_2 - 2s^*)$$

$$> (q_1 + q_2 - 2q_1) = (q_2 - q_1)$$

Therefore we conclude that if the snapshot s^* is placed before queries T_q, the cost to perform both queries is greater than when the snapshot is placed between the two queries.

Case 3: $s^* \notin [q_1, q_2]$ and $q_1 < q_2 < s^*$.

$$cost(T_q|s^*) = \sum_{i=1}^{2} |q_i - s^*| = (s^* - q_1 + s^* - q_2) = (2s^* - q_1 - q_2)$$

$$> (2q_2 - q_1 - q_2) = (q_2 - q_1)$$

hence, if the snapshot s^* is placed after the queries T_q, then the cost of performing those queries are greater than when the snapshot is placed between them.

From case1, case2 and case3, we can conclude that the optimal timestamp on the timeline where we can place the single snapshot $s^* \in \mathcal{T}$ to perform two queries $T_q = \{q_1, q_2 : q_i \in \mathcal{T}\}$, where $q_1 < q_2$ is when $s^* \in [q_1, q_2]$, meaning that $q_1 \leq s^* \leq q_2$, where the cost is equal to $q_2 - q_1$.

Now, we generalize our conclusion from the cases that we evaluated, for the placement of a single snapshot in the presence of n number of queries on the timeline:

Suppose that there is a set of queries $T_q = \{q_1, q_2, ..., q_n : q_i \in \mathcal{T}\}$ performed on the timeline. To evaluate the most optimal position to place the single snapshot $s^* \in \mathcal{T}$ for materialization, we breakdown the set of queries into the set of nested intervals $[q_1, q_n], [q_2, q_{n-1}], ..., [q_i, q_{n+1-i}]$ where n is the number of queries on timeline and $i = 0, 1, 2, ..., c$ where $c = \frac{n+1}{2}$ when there are odd number of queries and $c = \frac{n}{2}$ when there are even number of queries present on the timeline.

Based on the conclusion that we obtained from examining case 1, case 2 and case 3 earlier, for each nested interval, the cost of queries inside them is minimized if snapshot s^* is placed in a middle of the interval. Therefore if the snapshot is placed in a position which $s^* \in \{[q_1, q_n] \wedge [q_2, q_{n-1}] \wedge ... \wedge [q_i, q_{n+1-i}]\}$ the overall cost for all queries is minimized. In other words, if the snapshot is placed in a position that is in the middle of all nested intervals, then the total sum of absolute deviation of the snapshot from all queries is minimized. The placement of snapshot s^* in the median position of T_q, guarantees that the snapshot is placed in the middle of all nested query intervals, where the cost of queries is calculated as follows:

$$cost(T_q|s^*) = \sum_{i=1}^{n} |q_i - s^*| =$$

$$[(|q_1 - s^*| + |q_n - s^*|) + (|q_2 - s^*| + |q_{n-1} - s^*|) + ... + |q_c - s^*| + |q_{n+1-c} - s^*|)] =$$

$$[(s^* - q_1 + q_n - s^*) + (s^* - q_2 + q_{n-1} - s^*) + ... + (s^* - q_c + q_{n+1-c} - s^*)] =$$

$$[(q_n - q_1) + (q_{n-1} - q_2) + ... + (q_{n+1-c} - q_c)]$$

where parenthesis indicate the deviation from endpoints for one of nested intervals. In the case when there are odd number of queries performed on the timeline, the innermost interval is $[q_{\frac{n+1}{2}}, q_{\frac{n+1}{2}}]$ and the position of $q_{\frac{n+1}{2}}$ is the optimal position to place snapshot s^*. also when there are even number of queries the innermost interval is $[q_{\frac{n}{2}}, q_{\frac{n}{2}+1}]$, therefore if we choose snapshot s^*'s position to be at $q_{\frac{n}{2}} \leq s^* \leq q_{\frac{n}{2}+1}$,it guarantees that the snapshot exists inside each of nested intervals, and hence the sum of absolute deviation is minimized.

Optimal Materialization of Multiple Snapshots

Based on proposition x, the optimal placement of a single snapshot for multiple queries is a straight forward process, however in practice, the goal is to place an arbitrary m number of snapshots in order to lower the overall cost of queries. The constraint to choose m is determined based on the available resources.

Proposed in this thesis, a pattern could be obtained from performed query timestamps. Based on the pattern and with respect to the number of possible snapshots, an optimal non-overlapping segmentation of queries could be partitioned. As proved in proposition X, the median of query timestamps of each segmentation is the optimal position to place snapshots. Any queries that fall into the boundaries of a segmentation, use the respective snapshot of that segmentation for materialization.

Proposition (Segmentation of queries): Given an ordered set of snapshot timestamps $S = \{s_1, s_2, ..., s_m : s_i \in \mathcal{T}\}$, such that $s_i \leq s_{i+1}$, and n number of queries $Q = \{q_1, q_2, ..., q_n : q_i \in \mathcal{T}\}$, snapshots create m number of non-overlapping segments on the queries $Q[1, i_1], Q[i_1 + 1, i_2], ..., Q[i_{m-1}, i_m]$ such that queries in the segment $Q[i_j, i_{j+1}]$ use s_j to answer the queries in the optimal query answering strategy.

In this research, three different approaches were examined to optimally create query segmentations:

- Recursive Algorithm
- Dynamic Programming
- K-mean Clustering

Recursive algorithm and dynamic programming method guarantee the exact optimal query segmentation, while K-mean clustering approximates the optimal segments. In subsequent sections, each three methods are discussed, evaluated and compared.

Recursive Algorithm Formulation

Let $\text{opt}(Q, m)$ be the optimal m-snapshot placements for the query workload Q. Denote $Q[i, j] = \{q_i, q_{i+1}, ..., q_{j-1}, q_j\}$.

Proposition. (Optimality of sub-problems) Let $S^* = \text{opt}(Q, m)$. Let $\mathcal{Q}$ be the partition of segments created by S^*. Then, the prefix of S^* is also an optimal $m - 1$ snapshot placement of the prefix of $\mathcal{Q}$. Formally,

$$\text{prefix}(S^*) = \text{opt}(\cup\text{prefix}(\mathcal{Q}), m - 1)$$

We can formulate a recursive definition of $\text{opt}(Q, m)$ using Proposition **??**. The intuition is that we try out all possible *last* segment of Q, and pick the one with the lowest cost.

The recursive definition of $\text{opt}(Q, m)$ is given as:

- Base case $\text{opt}(Q, 1) = \{\text{median}(Q)\}$.
- Induction on m:

$$i^* = \text{argmin}\{\text{cost}(\text{opt}(Q[1, i], m - 1)) : i \in [1, n]\}$$

$$\text{opt}(Q, m) = \text{opt}(Q[1, i^*]) \cup \{\text{median}(Q[i^* + 1, n])$$

Proposition. The recursive formulation of $\text{opt}(Q, m)$ requires $\mathcal{O}(2^m)$.

Utilizing dynamic programming to find the optimal query segmentation has numbers of benefits and drawbacks:

Benefits:

- Precise optimal segmentation

Drawbacks:

- inefficiency in computation for large number of queries
- inefficiency in computation for large number of snapshots

Dynamic programming formulation

We can build a table **OPT** as a two dimensional array indexed by (i, k) where $i \in [1, n]$ and $k \in [1, m]$. Each entry in the table $\mathbf{OPT}[i, k] = \text{opt}(Q[1, i], k)$. We can compute $\mathbf{OPT}[i, k]$ in a bottom up fashion [].

```
computeOPT(Q, m) =
 n = |Q|
 OPT[i,0] = ∞
 for k = 1 → m
  for i = 1 → n
   j* = argmin_{j∈[1,i]} (cost(OPT[j,k−1]) + cost(Q[j+1,n]))
   OPT[i,k] = OPT[j*,k−1] ∪ {median(Q[j+1],n)}
  end for
 end for
```

proposition. The complexity of computing all the entries of **OPT** is $\mathcal{O}(mn^2)$.

While dynamic programming offers a fraction of time cost in comparison with recursive algorithm, but query segmentation in presence of large number of queries is still an overly-burden task. To summarize the benefits and drawbacks of this technique:

Benefits:

- Precise optimal segmentation

- Lower computation cost in comparison with dynamic programming

Drawbacks

- inefficiency in computation for large number of queries
- inefficiency in computation for large number of snapshots

K-mean Clustering

: Another solution to solve the optimal snapshot placement problem is to approximate the optimal query segmentation. For this purpose, we chose K-mean clustering algorithm.

Given a set of query timestamps $T_q = \{q_1, q_2, ..., q_n : q_i \in \mathcal{T}\}$, K-mean clustering method aims to predict k number of clusters $\mathcal{C} = \{c_1, c_2, ..., c_k\}$ from T_q, such that:

$$c_i = \arg\min_{\mathcal{C}} \sum_{i=1}^{k} \sum_{T_q \in c_i} ||T_q - \mu_i||^2$$

where μ_i is the mean of queries in c_i.

elbow method: one of the advantages of elbow method in K-mean clustering for this project is to find the effective m number of centroids.

To do so, we first initialize the cluster centroids $\mu_1, \mu_2, ..., \mu_m \in \mathcal{T}$ randomly. then we repeat the following algorithm until convergence.

for $i = 1 \rightarrow n$

$l^i = argmin_m ||q_i - \mu_m||$

for $j = 1 \rightarrow m$

$\mu_j = \frac{\sum_{i=1}^{m} 1\{l_i=j\} q_i}{\sum_{i=1}^{m} 1\{l_i=j\}}$

3.3 Applying cryptography

3.3.1 Signing

3.3.2 Signature validation

3.4 Blockchain

3.4.1 Creating Blocks

3.4.2 Block Validation

3.4.3 report fake data

Chapter 4

Experiments

Experiments here

Chapter 5

Related Work

Related works here

Chapter 6

Conclusions

Conclusion here

Bibliography

[1] Random, R. How random is everywhere. In *Proc. of the 2nd Work. of Randomness* (Apr. 2012), pp. 34 –41.