Towards Trusted and Efficient Temporal Relational Databases

By:

Mohammadamin Beirami

Supervisors:

Dr. Ken Pu

Dr. Ying Zhu

A thesis submitted to the

School of Graduate and Postdoctoral Studies in partial

fullfillment of the requirements for the degree of

Master of Science in Computer Science

Faculty of Science

University of Ontario Institute of Technology

Oshawa, Ontario, Canada

December, 2018

Thesis Examination Information

Submitted by: **Mohammadamin Beirami**

Master of Science in Computer Science

Thesis title: Towards Trusted and Efficient Temporal Relational Databases

An oral defense of this thesis took place on December 4, 2018 in front of the following examining committee:

Examining Committee:

Chair of Examining Committee: **Dr. Khalil El-khatib**

Research Supervisor: **Dr. Ken Pu**

Research Co-supervisor: **Dr. Ying Zhu**

Examining Committee Member: **Dr. Jaroslaw Szlichta**

External Examiner: **Dr. Min Dong**

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

Declaration of Authorship

I, Mohammadamin BEIRAMI, declare that this thesis titled, "Towards Trusted and Efficient Temporal Relational Databases" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: 

Date: December 14, 2018

Abstract

This research is to develop a blockchain based database based on a relational database management system (RDBMS). The database guarantees authenticity of the stored records using immutable and verifiable signatures. The database supports query evaluation over the entire timeline of the transaction log. To support efficient query evaluation, we utilize materialized snapshots at selected timestamps. We propose an optimal snapshot algorithm to compute the best timestamps so that the overall query load can be evaluated with minimal computation.

Acknowledgements

This thesis owes its existence to the help, support and inspiration of several people. Firstly, I would like to express my sincere gratitude to Dr. Pu and Dr. Zhu my advisors, for their supports, understanding, patience, enthusiasm and encouragement and for pushing me further than I thought I could go. I would also like to thank Dr. Szlichta and Dr. Dong for their assistance and suggestions to write this thesis. I wish to take this opportunity to express my sincere and particular appreciation to my family for their unflagging love, tremendous support and unconditional encouragement throughout my life and my studies.

Contents

List of Figures

List of Tables

Chapter 1

Motivation and Problem Definition

1.1 Motivation

Historical data is widely analysed for a lot of purposes such as making data-driven decisions [27]. Historical data could be financial reports, project data, emails, audit logs or any similar documents which contain past occurrences in an organization.

Historical data of a database provides data provenance which makes it possible to investigate the origin, the cause and the time of the processes that made changes to the tables and records ever exsisted in a database [3]. This data could be collected by auditing all the transactions that occur on a database system and store them in a logfile [12]. Although in large database systems, storing these historical data could seem burdensome and impractical due to the size they may get into [5], but current big data technologies such as cloud storages has made it possible to store these information in an easier and more affordable way than before [35]. In practice, not only these logfiles are useful assets that could be analyzed to detect maliciously manipulated data by outsider adversaries, but they can also give a valuable insight into the activities of the privileged users on a database system [31].

Utilizing logfiles as the source of data provenance gives us a useful tool to establish a trustworthy database environment [37], however this requires to make sure that the records in the logfile are trusted themselves [6].The challenge is that these files are not immune from being compromised [39][21]. In fact, not only the inherent security tools of most of the database management systems including the relational database management system (RDBMS) has proven to be vulnearable to complex cyber-security attacks [38], but also the malicious activities of the insiders who can bypass all the security requirements can add up to the complexity of the problem [39].

For a better understanding of the complexity of malicious attacks on the log files, consider a scenario in which a database super-user who acts as root can perform a lot of administrative tasks on a relational database. Since the super-user can bypass all the security requirements, they can remove the trace of their malicious activities from the log file. Such attempts results in the maliciously altered data in the database that are hard to identify because there are no evidences to contradict their legitimacy.

The afformentioned example clearly proves that restricting the activities of the users or utilizing preventive security tools for the database system cannot guarantee that the records will always remain trustworthy. Therefore, there is a need of a mechanism that makes the complexity of creating undetectable forgeries on the data provenance sources extremely expensive and unimaginable for all the users regardless of their level of accessibility to the system. The system should also be able to detect any malicious or accidental data manipulation on the database solely by relying on the verifiable evidences.

Our objective is to design a mechanism based on relational database management system that provides trustworthy logfiles that are used as the source of data provenance of the database system. To achieve the objective of both

providing trusted data and a functional system, we argue that the following requirements need to be addressed.

- ***Immutability of the records:*** To make sure that the a record in a relational database is trustworthy, performing undetectable malicious modification on that record should be extremely expensive. That is, any malicious or accidental attempts to modify the records stored in a database should result in an evident inconsistency in the data stored.

- ***Verifiability of trustworthiness:*** For all the records that are submitted in a database system, the origin of that transaction as well as the ligitimacy of the origin should be investigatbale using the verifiable evidences .

- ***Fast querying:*** The temporal relations are widely used for decision making, data analytics and more. Since the temporal relations could potentially become extremely large, a mechanism is needed to guarantee that the queries are answered in an acceptable time.

  The tables that contain historical records can become extremely large, and as a result, querying on these tables and verifying the trustworthiness of the result of queries can become inefficient [5][2]. For the system to be functional, a mechanism should be implemented in order to reduce the cost of both answeing to the queries and verifying the trustworthiness of the query results.

- ***Fast appending:*** The system needs to collect all the security information of the transactions and append them to the logfile in an immutable way. In order to create a functional system, this process should be done as cheap as possible.

Our research is a generic data model for temporal relations that can be applied to a wide range of scenarios. We believe that our proposed system is powerful in the sense that it removes the user-based trust and testifies the credibility or invalidity of the records by analyzing verifiable evidences. The system is able to detect privileged database misuses such as altering the auditing mechanism or manipulating the historical data.

We also made a contribution on performance aware optimization for large query workload on the temporal relations. This optimization is needed because the temporal relation can become extremely large in size and as a result the query latency could become high. By using the proposed method in this thesis, the select-query-aggregation queries on the temporal relation could be answered in a lower cost.

Our system utilizes many native to RDBMS tools such as relational temporal tables to store the historical data and cryptographic techniques to make transactions immutable and verifiable, therefore it could be supported by all RDBMSes available today with least additional requirements.

Chapter 2

Background and Related Work

In this chapter, the concepts and definitions which are needed to implement the proposed system are identified. In the first section, the tools that are needed to build a blockchain-based strategy are introduced. In the next section, the tools and concepts borrowed from the relational database management system are identified. The last section is also dedicated to discuss the related work and the researches which have been done in this field.

2.1 Blockchain

Blockchain technology is a distributed trusted public ledger that the stored data in it are linked using cryptography. The stored data in Blockchain are immutable and open to anyone to inspect [25]. There are a numerous number of applications that are using Blockchain [8], however, it got its reputation because of its application as the backbone of cryptocurrencies such as Bitcoin [24]. We utilize Blockchain technology because it is able to provide immutability and verification to the stored records in a temporal table. The immutability that the Bockchain provides makes forgeries on the records extremely difficult. Also it enables every user of the system to investigate about the trustworthiness of the

records without revealing the confidential information. In this section, the ingredients of creating a Blockchain is discussed.

2.1.1 Digital signatures

Digital signatures are the main building block of Blockchains. Blockchain utilizes digital signatures for the purpose of providing immutability and verification for its stored data. The ingredients of building a digital signatures are cryptographic keys, asymmetric encryption and hash functions that we discuss them in this section.

Definition 1 (Cryptographic Keys) *Cryptographic keys* $\langle K_{priv}, K_{pub} \rangle \in \mathbf{N}^+$ *are a pair of strings that are generated using mathematical functions, 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 are transmitted between the users[34].*

The procedure of creating a pair of Cryptographic keys is shown in Algorithm 1.

Function `generateKeys(`*keySize*`)`**:**
    $randomVal \leftarrow Random.new()$
    $publicKey, privateKey = RSA.generate(keySize, randomVal)$
    return *publicKey*,*privateKey*

Algorithm 1: Generate Cryptographic keys

The cryptographic keys are mainly used to encrypt data. The purppose of encryption is to convert data from their ordinary form to unintellegible information that are unreadable by unprivileged users [16]. By having a pair of cryptographic keys $\langle K_{priv}, K_{pub} \rangle$, the encryption is done asymmetrically.

Definition 2 (Asymmetric Encryption) *Given the cryptographic keys* $\langle K_{pub}, K_{priv} \rangle$ *and a message m, an encryption technique is said to be asymmetric if:*

$$c = Encrypt(K_{pub}, m) \text{ and } m = Decrypt(K_{priv}, c)$$

or

$$c = Encrypt(K_{priv}, m) \text{ and } m = Decrypt(K_{pub}, c)$$

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} [34]. Algorithm 2 shows the basic steps to encrypt a message using K_{pub} and Algorithm 3 shows the steps to decrypt a message using K_{priv}

Function `encrypt(`m,K_{pub}`)`**:**
    publicKeyObj = RSA.importKey(K_{pub})
    $randomParam \leftarrow random.choice()$
    return publicKeyObj.encrypt(m,$randomparam$)

Algorithm 2: Encrypt a message using public key

Function `decrypt(`$enc\_message$,K_{priv}`)`**:**
    privateKeyObj = RSA.importKey(K_{priv})
    return privateKeyObj.decrypt($enc\_message$)

Algorithm 3: Decrypt an encrypted message using private key

In addition to asymmetric encryption, hash functions are also another ingredient of the digital signatures.

Definition 3 (Hash function) *Assume m to be the message with an arbitrary size chosen from domain* $\mathcal{A}$. $hash(m) \rightarrow$ *sketch is a function that maps the m of any size from domain* $\mathcal{A}$ *to a fixed size string (normally 256 bits) in a smaller domain* $\mathcal{B}$ *[1].*

Now we discuss the procedure of generating a digital signature using the tools that we introduced.

Definition 4 (Digital signature) *Let m be the message which needs to be digitally signed. In order to digitally sign the message m using the private key K_{priv} the steps shown in Algorithm 4 are taken:*

Function `digitalSignature`(m,K_{priv})**:**
    $hashVal$ = hash(m)
    $randomVal$ = random.choice()
    $privateKeyObj$ = RSA.importKey(K_{priv})
    return $privateKeyObj$.encrypt($hashVal$,$randomVal$)

Algorithm 4: Creating the digital signature of m

Note that only the person who has the private key is able to create digital signatures. On contrary, anyone who has access the public key of the users are able to verify the digital signature.

Definition 5 (digital signature verification) *given a digital signature "signature", the message m and the public key K_{pub}, the steps that needs to be taken in order to verify the authenticity of the record using a digital signature is shown in Algorithm 5:*

Function `verifySign`(m,$signature$,K_{pub})**:**
    $publicKeyObj$ = RSA.importKey(K_{pub})
    $hashVal$ = hash(m)
    $signHash$ = publicKeyObj.decrypt($signature$)
    if $hashVal$ == $signHash$ **then**
        return $valid$
    return $invalid$

Algorithm 5: Verify the authenticity of the record using digital signature

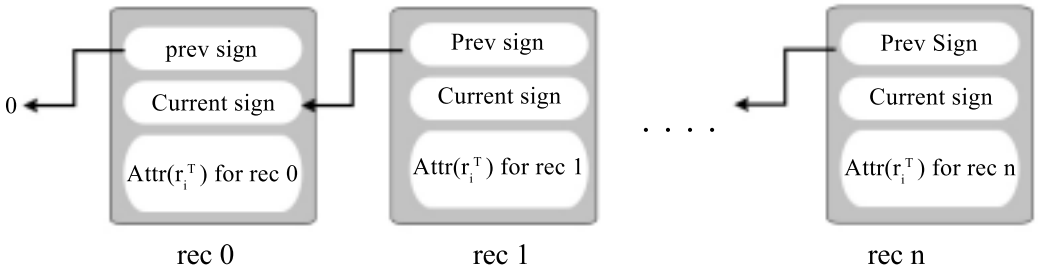

FIGURE 2.1: Idea of the Blockchain for the proposed system.

As mentioned earlier, digital signatures are the main ingredient of a Blockchain. In the follwing, we discuss the creation of Blockchain for a database relation, using digital signatures.

Definition 6 (Blockchain) *Let rec_i be specific records stored in a relation r^T. We denote sign as the digital signature of each record. We can create a blockchain of multiple records by augmenting an attribute prev_sign in rec_i that stores the digital signature of rec_{i-1} in rec_i.*

The idea of the Blockchain has been depicted in figure 2.1.

2.2 Temporal Relational Databases

The main focus of this research is to build a trusted temporal relation using Blockchain technology that provides data provenance for the database system. In this section, we introduce the tools and concepts that were utilized in this research to establish a trusted temporal relation.

Definition 7 (Temporal database) *Let r, be a relation in the database D. Denote the attributes of the relation as $attr(r)$. A temporal table of relation denoted r^T is a table with attributes $attr(r^T) = attr(r_i) \cup \{timestamp, deleted\}$ where timestamp is the time in which transactions on r happened and deleted is a flag showing whether or not*

that transaction was meant to delete a record from r. A temporal database denoted D^T is the result of augmenting D by r^T:

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

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

Example 1 *Given a normal relational table r_1 (Table 2.1) and a temporal table r^T (Table 2.2), the $attr(r_1) = \{id, item, value\}$ and $attr(r^T) = \{id, item, value, updated, deleted\}$. The result of a few example queries are:*

- *q_1: SELECT * FROM r WHERE id = 22;*
  Result: $[(22, Pencil, 7.50)]$
- *q_2: SELECT * FROM r^T WHERE id = 22;*
  Result: $[(22, pencil, 8.0, 2018-03-21, False),$
  $(22, pencil, 7.50, 2018-03-30, False)]$
- *q_3: SELECT * FROM r WHERE id = 21;*
  Result: $[]$
- *q_4: SELECT * FROM r^T WHERE id = 21;*
  Result: $[(21, ruler, 3.25, 2018-02-10, False),$
  $(21, ruler, 3.25, 2018-02-20, True)]$

This example clearly shows that the temporal tables can provide data provenance for the records ever existed in the normal relations. For exmaple by having the temporal table "r^T" it could be seen that the record with "$id = 22$" used to have the value of "8.0" set in "$2018-03-21$" but then changed to "7.50" at "$2018-03-30$". Also, once there was a record with "$id = 21$" but deleted at "$2018-02-20$" and that is why the query q_3 does not return any results.

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^T

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

The temporal database provides information about the timestamp in which the transactions on the database system occurred.

Definition 8 (Time domain) *The time domain $\mathcal{T}$ consists of discrete timestamps $t_0, t_1, ..., t_n$ in which transactions on tables $r \in D$ happened. The range of time domain is: $\mathcal{T} = [t_0, t_n]$ where the lower bound t_0 is the timestamp in which the first record added and the upper bound t_n is the timestamp of the most recent transaction to the table r.*

Example 2 *The time domain of a temporal table r^T (Table 2.2) is given by:*

$$\mathcal{T} = range(r^T[timestamp]) = [2018-02-10, 2018-04-01]$$

Definition 9 (Timestamps) *A timestamp $t_i \in \mathcal{T}$ is a particular position in the time domain, in which (a) particular transaction(s) happened. For example in the temporal table r^T 2.2, "2018-03-30" is a timestamp in which the record with "id = 22" updated.*

Definition 10 (Timeline) *Let $u_1, u_2, ..., u_n$ be the total number of transactions on the tables $r \in D$ at timestamps $t_j \in \mathcal{T}$, where $j = \{0, 1, ..., n\}$. These transactions could*

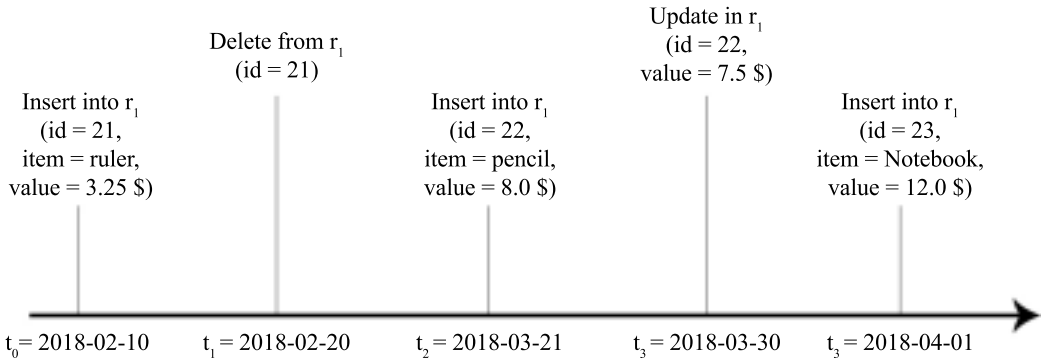

FIGURE 2.2: Timeline.

be represented as an ordered set points on a vector. This vector is called the timeline of transactions for $r \in D$.

Figure 2.2 illustrates the concept of timeline.

Given a temporal database, for the sake of data analytics, it is a common practice to query for the form of a table in an specific timestamp. These queries could be answered by creating snapshot of the relation using the temporal table.

Definition 11 (Snapshot) *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* $\mathcal{T} \leq t$. *A snapshot is a materialized version of* $D(t) = \{s_1(t), s_2(t), ..., s_n(t)\}$.

Example 3 *Given a normal relational table* r_1 *(Table 2.3), the temporal relational table* r^T *(Table 2.4) contains the historical data of* r_1. *The snapshot of the* r_1 *at* $t = 2018 - 04 - 01$ *could be generated as Table 2.5.*

TABLE 2.3: Normal Relational Table r_1

id	item	value
22	Pencil	7.50
23	Notebook	12.0
24	Console	230.0

TABLE 2.4: Temporal Table r^T

id	item	value	timestamp	deleted
21	Ruler	3.25	2018-02-10	False
22	Pencil	8.0	2018-03-21	False
22	Pencil	9.0	2018-03-30	False
23	Notebook	11.0	2018-04-01	False
22	Pencil	6.0	2018-04-01	False
21	Ruler	3.25	2018-04-02	True
23	Notebook	12.0	2018-04-02	False
22	Pencil	7.50	2018-04-05	False
24	Console	230.0	2018-04-05	False

TABLE 2.5: Normal Relational Table r_1 at t = 2018-04-01

id	item	value
21	Ruler	3.25
22	Pencil	6.0
23	Notebook	11.0

2.3 Related Work

There has been a wide range of studies on ensuring the trustworthiness of records in a database from different perspectives. These studies range from establishing trust between nodes in a real-time distributed systems [17] to secret sharing schemes in cloud databases [10] and utilizing logfiles for forensic purposes [31]. In this project, the assumption is that the preventive models are not able to stop the adversaries from manipulating the data of a relational database system, therefore a tamper-evident log table has been offered to evaluate whether or not the data has been altered.

Database audit logs contain valuable information such as any insertions, deletions, and modifications of the records performed in a database along with the timestamp of the performed tasks. The United States Department of Defense in its "Trusted Computer System Evaluation Criteria" document and under requirement 4 pointed out the importance of auditing the transactions in a computer system in a secure and efficient manner [7]. In this document also protecting the audit logs from modification or destruction has been stressed out.

Analyzing audit logs for forensic purposes has been the topic of research by many scientists, however, since the trustworthiness of the results from log table analysis has a direct relationship with the authenticity of the records of the log file, a lot of studies have been carried out to make log tables tamper-proof. Haber *et al.*[13] proposed a basic methodology by utilizing timestamping and hash chains in order to make unmodifiable historical records for digital documents. Peha in [26] offered a method to detect log tampering by one way hashing and employing multiple trusted notaries to keep track of all transactions. The author argued that if any notary decided to falsify the transaction, the attempt is discovered by other notaries. Snodgrass *et al.* [32] also offered

one-way hashing mechanism and employed trusted notaries, however in order to enhance the security of the method offered by Peha, they offered to hash the records with a timestamp of previous transaction modification. Schneier *et al.*[28] offered a cryptographic-based mechanism to create hash chains and make the log files nearly impossible for the attacker to alter. The validity of the transactions was also done by trusted third-parties who have the cryptographic keys.

However the aforementioned researches might have promising results to protect the historical records from being compromised by an outsider, but they have one thing in common which is the role of an insider to carry out malicious attacks is forgotten. Also hiring third-party software/hardware may bring up a lot of privacy concerns. Therefore, unlike the mentioned works, not only our proposed system does not require a third-party notary to attest the authenticity but also it does not put trust on any user of the system.

The role of privileged users in acting maliciously in a database has been discussed by many researchers [5] [39]. Liu *et al.* [22] offered a network-based auditing mechanism which also keeps track of privileged users' activity and performs audit analysis through event correlation. Wagner *et al.* [38] proposed a mechanism to detect database file tampering by looking for inconsistencies in the database's storage. The authors argued that the databases follow patterns in storage which even the privileged users have no access to. Therefore, if a record is deleted maliciously in the log file, the inconsistency in the storage is evident for a period of time. Unlike mentioned proposed methods, our system uses inherent to RDBMS tools and does not require a network-level-auditing mechanism or having access to the server's storages, therefore it could be easily implemented on remote servers and relational databases on cloud storages. Also, our proposed system not only discovers maliciously deletion of the records but also identifies any malicious modification without any time constraints.

For the sake of gathering verifiable pieces of evidence, in our proposed system, any changes to the database regardless of the users' access privileges need to be tracked. This action could be done by utilizing inherent to DBMS tools. Fabbri *et al.* [11] extensively talked about SELECT triggers which are inherent RDBMS functions, required database specifications and efficient implementation techniques for data auditing. Hauger *et al.*[14] also discussed the use of triggers in the database for forensic purposes. Triggers are supported by RDBMS which makes them a good candidate to be used in our proposed system however, it is naive to assume that a simple trigger solely is secure enough to be used for forensic purposes. Therefore we propose to use the Blockchain technology to make immutable chains of transactions that are captured by database triggers.

The first attempt to use chained hashes for securing data from tampering is known to be done by Haber *et al.* [13] where the authors proposed a methodology to securely timestamp the digital files and create digital signatures and hash chains. This work was then improved by Schneier *et al.* [28] [29] [30] by offering to exchange secret codes with a trusted third party who is able to verify the authenticity of the chain. They offered a method to change the shared secret as the new transaction occurs, therefore since the attackers do not have the previous secret codes, it is impossible for them to alter any records which were previously added. Our proposed method is similar to the mentioned works in this aspect that our historical records are chained together using the digital signatures, however, we use asymmetric encryption to generate digital signatures that removes the need of the third-parties to attest the authenticity of the transactions. The asymmetric encryption method also enables anybody to verify signatures without revealing the users' secrets.

Chapter 3

Problem Definition and Algorithms

In this chapter, we introduce a practical procedure to implement the Blockchain methodology in a temporal relational database as well as the principles to verify trustworthiness of the records and Blockchain. Then we talk about creating snapshots of relations using the temporal database. In that section we fomally define the problem of large query workload inefficiency in the context of temporal relational databases and then we propose a solution to lower the cost of such queries. Majority of the definitions, figures, examples and discussions used for the topic of snapshot materialization of this chapter were borrowed from our paper in [2].

3.1 Development of Blockchain-based temporal relations

In this section we extensively talk about the implementation of Blockchain for the temporal relational database. The first step towards this objective is to augment some attributes to the relations that hold the security information of the records.

3.1.1 Temporal table with security information

The first step towards building a temporal table with security information is to generate a pair of Cryptographic keys $\langle K^u_{priv}, K^u_{pub} \rangle$ and a unique username for each user of the system. Using the Cryptographic keys, the system is able to create and verify digital signatures of the transactions performed by users. We previousely discussed the generation of digital signatures in 2.1.1. Now we talk about the generation of digital signatures in the context of the proposed temporal database.

Definition 12 (digital signature of the transactions) *Let u be the user who submitted a set of records $rec_j = \{rec_1, rec_2, ..., rec_n\}$ to r^T_i. Given the private key of the user denoted as K^u_{priv} the signature of each record could be computed as*

$$signature(rec_j | K^u_{priv}) = encrypt(hash(rec_j), K^u_{priv})$$

By storing the digital signature of the records, we provide trustworthy verification for the records. This attempt makes it difficult for the adversaries to forge the records of the database, however the undetectable malicious activities are still possible. In order to add to the difficulty of forgeries on the temporal relation, we tend to create a chain of records using the digital signatures.

Definition 13 (Temporal table with Blockchain) *The temporal relational table with chained security information denoted as r^{T*}_i is a table with the attributes*

$$attr(r^{T*}_i) = attr(r^T_i) \cup \{username, currentSignature, previousSignature)\}$$

TABLE 3.1: Temporal Table r_1^T with chained security information

r_id	id	item	value	timestamp	deleted	username	curSgn	prevSgn
151	21	Ruler	3.25	2018-02-10	False	Bob	r3T49TR	0
152	21	Ruler	3.25	2018-02-20	True	Alice	yu0PmER	r3T49TR
153	22	Pencil	8.0	2018-03-21	False	Alice	gI90vjN	yu0PmER
154	23	Pen	12.0	2018-03-30	False	Bob	89Ec578	gI90vjN
155	22	Pencil	7.50	2018-04-01	False	Eve	Ipu32h6	89Ec578

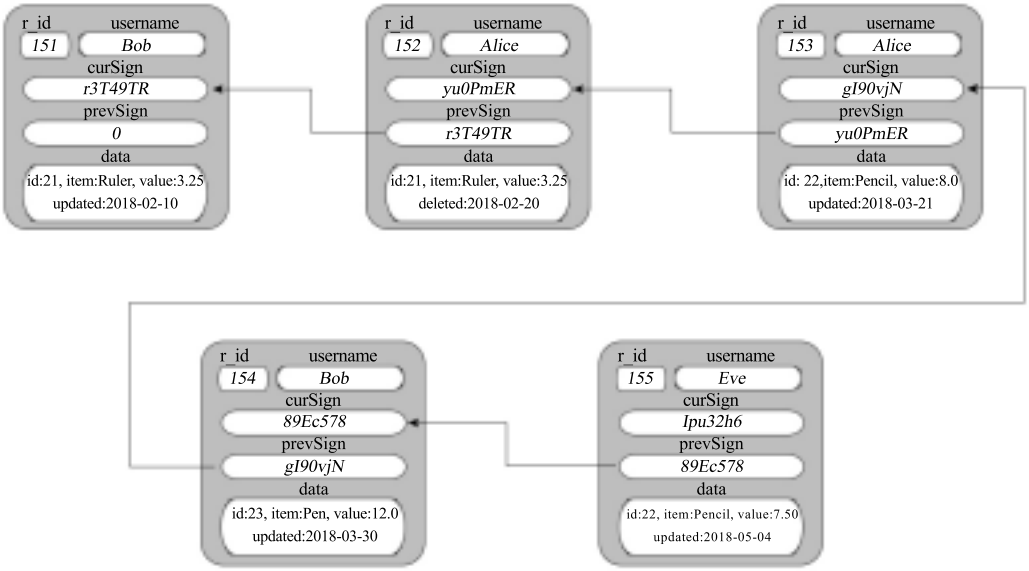

FIGURE 3.1: Blockchain representation of the table r_1^T.

where username is the unique identity of the user who submitted the transaction, currentSignature is the digital signature of the submitted transaction and previousSignature is the signature of the previous record stored in r_i^T.

Example 4 *The Table 3.1 is an example of a temporal relational table with chained security information. With assumption that each record is a block, this table also could be depicted as Figure 3.1 which is the blockchian representation of Table 3.1.*

Definition 14 (Chain verification) *To verify if a chain of the records are valid, the following steps are proposed:*

- ***step 1.*** *verify the currentSignature of individual records.*
- ***step 2.*** *check if* $rec_j[previousSignature] == rec_{j-1}[currentSignature]$ *except for* rec_0

A chain is said to be broken if inconsistent information being gained in any of the above steps.

3.1.2 Appending transactions

We previousely talked about how to provide security information for the records of the temporal relations. In order to append a transaction along with its security information to these temporal relations, the following steps should be taken:

- First the triggers of the database detect transactions on the records of the database.
- The transaction submitter's information is retrived from the users table.
- The position of the updated record on the temporal table is determined and the digital signature of its previous record is retrieved.
- A digital signature of the updated record concatinated with its previous record's digital signature is computed by utilizing the transaction submitter's cryptographic keys.
- The updated record is appended at the end of the table and chained to the end of blockchain by pointing to its previous record's digital signature.

incremental manitenance of blockchain cost

By assuming that the updates δD on the temporal database occur in constant time, appending these updates to the end of blockchain is done in a constant time $\mathcal{O}(\delta D)$.

It is not realistic to assume that all the transactions in a database system occur in separate timestamps. In fact when concurrent transactions are performed, a database system without concurrency control, has no preference to decide which transaction to be submitted first, therefore it may result in an inconsistent database and consequently an inconsistent blockchain.

To solve the issue of concurrent transactions on the database, the transactions on the database need to be *serializable*. Serializability grants some order to the execution of concurrent transactions on the database system. This could be achieved by utilizing either the inherent to relational database engine concurrency control tools or software-based alternatives such as software transactional memory. In this project we utilized *serializable* isolation level that is defined by SQL standards and supported by Postgresql database to provide serializability for the concurrent transactions on the database system.

3.2 Snapshot creation

Our proposed trusted temporal database provides data provenance for the records that are stored in a relational database. Given a temporal relation, for the sake of decision making and data analytics, it is common to query for the version of a relation in an specific timestamp. These queries could be answered by creating snapshot of the relation using temporal database.

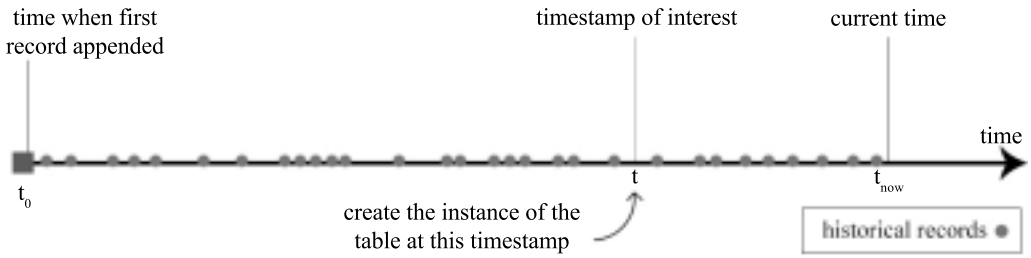

FIGURE 3.2: The notion of creating snapshot on the timeline.

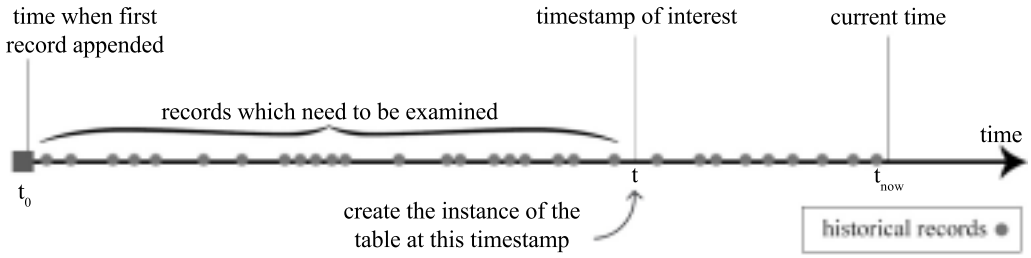

FIGURE 3.3: The records which needs to be checked when creating a snapshot.

Linear time in creating snapshots

Given a relation r, we are interested to create a snapshot of this relation at time t, using the temporal relation r^T. Assume that the table was updated at a constant rate over time, then the complexity of snapshot(r, t) is

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

Creating snapshots from timeline perspective could be depicted in Figure 3.2. Also the records that need to evaluated to create a snapshot can be shown in Figure 3.3. This clearly shows that computing a snapshot requires visiting each record that has been submitted before t once, and apply the updates. The problem is that, as the number of records to be evaluated grow, creating snapshot become computationally more expensive.

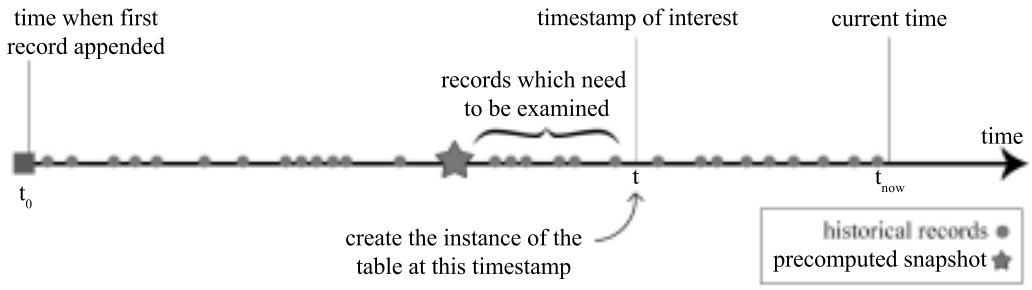

FIGURE 3.4: The records which needs to be checked when creating a snapshot with having a precomputed snapshot for materialization.

3.2.1 Query answering using materialized snapshots

Using pre-computed materialized view has been proven to be effective in reducing the computational time of query answering [33] [9]. As discussed earlier, running queries $Q(t)$ on temporal table r^T to build snapshots and verifying its Blockchain requires linear time with time complexity of approximately $\mathcal{O}(t)$. Consequently in the presence of multiple and concurrent queries, such transactions are computationally expensive and inefficient. We argue that, if a snapshot is computed at an specific timestamp t and placed on the timeline for materialization, the computational time of answering to the subsequent queries on r^T is reduced. The notion of having precomputed snapshots for materialization can be depicted as Figure 3.4. Note that the snapshot could exist before or after a query and in both cases, the query can use the snapshot for materialization.

Proposition 1 *suppose we have a precomputed snapshot s that is placed on the timeline for materialization. Then creating $snapshot(r, t)$ can be computed with complexity:*

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

where t is the timestamp of interest.

TABLE 3.2: Temporal Table r_1^T

id	item	value	timestamp	deleted
1	Paper	0.25	2018-02-10	False
2	Scissors	8.0	2018-02-12	False
3	Folder	1.50	2018-02-12	False
1	Paper	0.30	2018-02-13	False
4	Pencil	3.0	2018-02-16	False
3	Folder	1.75	2018-02-21	False
5	Batteries	8.0	2018-02-23	False
1	Paper	0.35	2018-02-25	False
6	Notebook	7.0	2018-03-01	False
5	Batteries	9.0	2018-03-01	False
4	Pencil	3.25	2018-03-04	False
1	Paper	0.35	2018-03-04	True
7	Ruler	4.0	2018-03-06	False
2	Scissors	8.50	2018-03-07	False
7	Ruler	4.50	2018-03-08	False
5	Batteries	11.0	2018-03-10	False
3	Folder	1.75	2018-03-11	True
7	Ruler	4.50	2018-03-12	True
6	Notebook	7.50	2018-03-15	False
2	Scissors	7.50	2018-03-17	False

Example 5 *Given a temporal relational table r_1^T (Table 3.2) and a precomputed snapshot table s_1 at timestamp $t = 2018-03-11$(Table 3.3), we are interested to create a snapshot s_2 which is the instance of table r_1 at the timestamp of $t = 2018-03-15$. In order to compute snapshot s_2, Table 3.4 shows the transactions which needs to be evaluated without using snapshot s_1 and Table 3.4 is when s_1 is used for materialization. This example clearly shows that when creating snapshot s_2 (Table 3.6), less transactions need to be evaluated when s_1 is used for materialization.*

TABLE 3.3: Snapshot s_1 at $t = 2018 - 03 - 11$

id	item	value
2	Scissors	8.5
4	Pencil	3.25
5	Batteries	11.0
6	Notebook	7.0
7	Ruler	4.50

TABLE 3.4: the transactions to compute snapshot s_2 at $t = 2018 - 03 - 15$ without using snapshot s_1 for materialization

id	item	Transaction	timestamp	value	query on
1	Paper	cretaed	2018-02-10	0.25	r_1^T
	Paper	updated	2018-02-13	0.30	r_1^T
	Paper	updated	2018-02-25	0.35	r_1^T
	Paper	deleted	2018-03-04	-	r_1^T
2	Scissors	cretaed	2018-02-12	8.0	r_1^T
	Scissors	updated	2018-03-07	8.50	r_1^T
3	Folder	cretaed	2018-02-12	1.50	r_1^T
	Folder	updated	2018-02-21	1.75	r_1^T
	Folder	deleted	2018-03-11	-	r_1^T
4	Pencil	cretaed	2018-02-16	3.0	r_1^T
	Pencil	updated	2018-03-04	3.25	r_1^T
5	Batteries	cretaed	2018-02-23	8.0	r_1^T
	Batteries	updated	2018-03-01	9.0	r_1^T
	Batteries	updated	2018-03-10	11.0	r_1^T
6	Notebook	cretaed	2018-03-01	7.0	r_1^T
	Notebook	updated	2018-03-15	7.50	r_1^T
7	Ruler	cretaed	2018-03-06	4.0	r_1^T
	Ruler	updated	2018-03-08	4.50	r_1^T
	Ruler	deleted	2018-03-12	-	r_1^T

TABLE 3.5: the transactions to compute snapshot s_2 at $t = 2018 - 03 - 15$ using snapshot s_1 for materialization

id	item	Transaction	timestamp	value	query on
2	Scissors	-	-	8.50	s_1
4	Pencil	-	-	3.25	s_1
5	Batteries	-	-	11.0	s_1
6	Notebook	-	-	7.0	s_1
	Notebook	updated	2018-03-15	7.50	r_1^T
7	Ruler	-	-	4.50	s_1
	Ruler	deleted	2018-03-12	-	r_1^T

TABLE 3.6: Snapshot s_2 at $t = 2018 - 03 - 15$

id	item	value
2	Scissors	8.50
4	Pencil	3.25
5	Batteries	11.0
6	Notebook	7.50

3.3 Single snapshot for materialization

Let $T_q = \{q_1, q_2, \ldots, q_n\}$ be the timestamps of n queries, each querying the database at $D^T(q_i)$. To save on computational cost in answering the queries on temporal database D^T, we propose to compute snapshot s in optimal timestamp on the timeline to answer to T_q at lower cost.

Definition 15 (Cost of Query Answering with single materialized snapshot) *In the presence of a single materialized precomputed snapshot s , the cost of answering the query T_q is calculated as:*

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

Definition 16 (Optimal Snapshot placement) *: For the* single snapshot placement *problem, the goal is to find the timestamp s^* such that*

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

3.3.1 Optimal single snapshot placement

Let $T_q^* = \{q_1, q_2, \ldots, q_n\}$ be query workload on the temporal database. T_q^* gives us a valuable insight into the query patterns on the temporal database, that

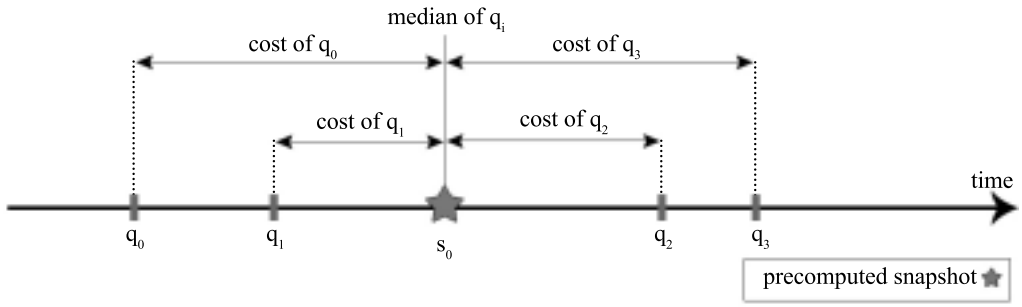

FIGURE 3.5: Placing a single snapshot in the median of queries guarantees the optimal cost of query answering.

could be used to find optimal position of snapshots.

Proposition 2 *Given the performed queries T_q^*, the optimal position for a single snapshot on the timeline for materialization is $s^*(T_q^*) = median(T_q^*)$ that can be computed in $\mathcal{O}(|T_q^*|)$.*

Figure 3.5 shows the notion of placing snapshot in the median of queries for materialization.

Proof of the Proposition 2: At first, we solve the problem of a single snapshot placement for two queries, and then we generalize the conclusion for multiple queries:

Assume that there are two queries $T_q = \{q_1, q_2\}$ on the timeline, such that, $q_1 < q_2$. for the placement of a single snapshot s^* 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 case 1, we can infer that the cost of running two queries q_1 and q_2 when the 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 that we can place the single snapshot s^* to perform two queries $T_q = \{q_1, q_2\}$, where $q_1 < q_2$ is when $s^* \in [q_1, q_2]$.

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

Suppose that there is a set of queries $T_q = \{q_1, q_2, ..., q_n\}$ performed on the timeline. To evaluate the most optimal position to place the single snapshot s^* 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 $c = \frac{n+1}{2}$ for odd number of queries and $c = \frac{n}{2}$ for even number of queries.

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.

3.4 Multiple snapshot placement

In the presence of thousands of queries on a temporal database with millions of records, having a single snapshot reduces the cost of query answering but it is still insufficient. Therefore, to reduce the overall cost of query answring, optimal timestamps should be computed on the timeline of the temporal database to place snapshots for materialization. In the following section, different approaches to compute the optimal positions on the timeline are discussed.

Proposition 3 (Cost of query answering with multiple materialized snapshots) *If multiple snapshots* $S = \{s_1, s_2, \ldots, s_m\}$ *were precomputed and materialized, then*

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

Proposition 4 (optimal multiple snapshots for materialization) *The* m-snapshot placement *problem is to compute* m *number of timestamps* $S^* = \{s_1, s_2, \ldots, s_m\}$ *to place* m *number of snapshots for materialization, such that*

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

Proposition 5 (Optimal number of snapshots) *Let* R *be the total resources available on the system specified by the system designer to handle the given workload, and* $\mathcal{L}$ *to be the average snapshot size. the maximum number of snapshots* N *could be determined by:*

$$N = R/\mathcal{L}$$

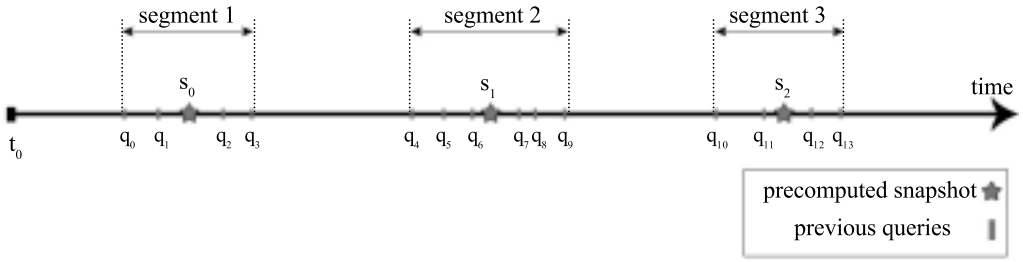

FIGURE 3.6: The notion of creating segmentations on the timeline and placing snapshot for each segmentation.

Now, we examine different approaches to find the optimal timestamps to place snapshots. 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 6 (Segmentation of queries) *Given an ordered set of snapshot timestamps $S = \{s_1, s_2, ..., s_m\}$, such that $s_i \leq s_{i+1}$, and a query workload $Q = \{q_1, q_2, ..., q_n\}$, 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.*

The notion of creating segmentations is depicted in Figure 3.6.

Proposition 7 (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,*

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

3.4.1 recursive approach to find optimal segmentations of the timeline

We can formulate a recursive definition of $\text{opt}(Q, m)$ using Proposition 7. 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:

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

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

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

The recursive algorithm could be written as Algorithm 6

Function `computeOPT(`Q, m`)`**:**
  $n = |Q|$
  $\mathbf{OPT}[i, 0] = \infty$
  for $k \leftarrow 1$ **to** m **do**
    for $i \leftarrow 1$ **to** n **do**
      $j^* = \underset{j \in [1,i]}{\text{argmin}}(\text{cost}(\mathbf{OPT}[j, k-1]) + \text{cost}(Q[j+1, n]))$
      $\mathbf{OPT}[i, k] = \mathbf{OPT}[j^*, k-1] \cup \{\text{median}(Q[j+1], n)\}$
    end
  end

Algorithm 6: Recursive algorithm method to compute m number of optimal segmentations

Example 6 *Given the queries* $Q = \{q_1 = 2, q_2 = 4, q_3 = 9, q_4 = 11, q_5 = 17, q_6 = 20\}$ *(Figure 3.7) and maximum number of snapshots* $m = 3$*, the goal is to find the most*

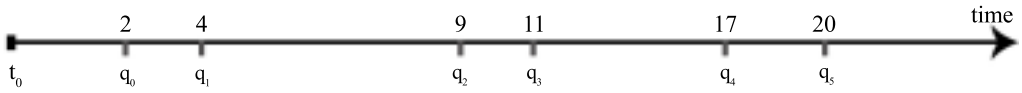

FIGURE 3.7: The quer on the timeline in Example.

optimal segmentations using recursive algorithm to place snapshots for materialization. The steps to compute m number of optimal segmentations is shown in Figure 3.8. In this example, the most optimal segmentation possible for 3 snapshots is when $segment_1 = \{q_0, q_1\}$, $segment_2 = \{q_2, q_3\}$, $segment_3 = \{q_4, q_5\}$ *where the total cost is 7 units. Figure 3.9.*

3.4.2 Dynamic programming approach to find optimal segmentation of the timeline

Dynamic programming improves the time complexity of finding optimal segmentations in recursive algorithm by utilizing memoization technique [15]. In the memoization technique, if the cost of a segmentation is calculated, it is stored in a table where the recursive calls can look up the results in the table instead of recalculating them. 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 [19]. The complexity of computing all the entries of **OPT** is $\mathcal{O}(mn^2)$.

The Dynamic programming could be implemented as Algorithm 7

Example 7 *Given a query workload* $Q = q_0 = 2, q_1 = 4, q_2 = 9, q_3 = 11$, *we want to compute 3 segmentations from these queries, such that putting a snapshot in each segmentation, make the overall cost of answering to the queries optimal. The process of computing optimal snapshots is shown in Table 3.7. This method is more efficient than recursive algorithm as for example in the memoization table T[3][2], the recursive*

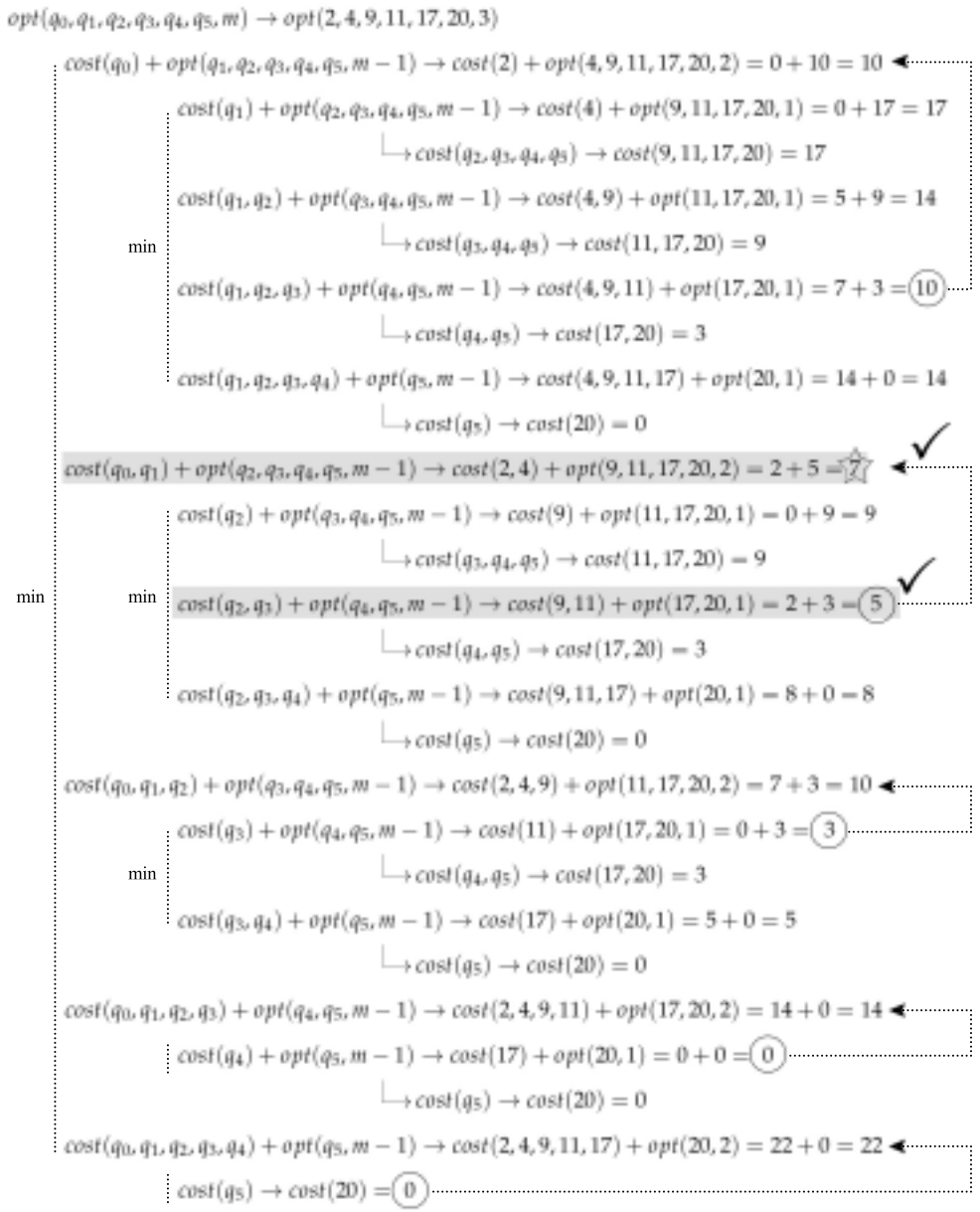

FIGURE 3.8: Recursive approach to compute optimal segmentations for 3 snapshots.

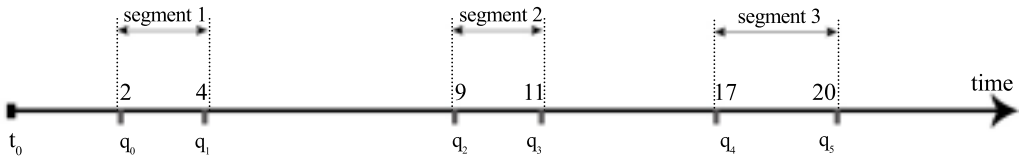

FIGURE 3.9: Segmentation of queries on the timeline for Example 6

Function `computeOPT`(Q, m)**:**
  $n = |Q|$
  $minVal = \infty$
  for $i \leftarrow 1$ **to** m **do**
    for $j \leftarrow 1$ **to** $n + 1$ **do**
      for $k \leftarrow 1$ **to** j **do**
        $minVal = \min(\text{minVal}, \textbf{Table}[i, k] + \text{cost}(Q[j - k :]))$
      end
      $\textbf{Table}[i, j] = minVal$
    end
  end

Algorithm 7: Dynamic programming method to compute m number of optimal segmentations

call uses the cost stored in T[2][1], T[2][2] and T[2][3] without the need to recalulate them. In this example, the most optimal overall cost is 2 which is achieved by either $\{segment1 = [q_0, q_1], segment2 = [q_2], segment3 = [q_3]\}$ *or* $\{segment1 = [q_0], segment2 = [q_1], segment3 = [q_2, q_3]\}$

3.4.3 Heuristic method to find optimal timeline segmentation

In the heuristic technique, the optimal solution to a problem is not guaranteed however it could be seen as a suitable alternative solution when runtime has more priority than the accuracy of the solution. For the purpose of finding the optimal segmentation of queries for optimal query answering, we utilized K-means clustering technique. We chose K-means clustering over other clustering methods because each centroids resulted from this technique are in the middle

TABLE 3.7: Memoization table T for dynamic programming approach to compute 3 optimal segmentations from 4 queries.

0	0	0	0	0
0	$T[0][0] + cost(q_0)$ $0+0=0$	$T[0][1] + cost(q_0, q_1)$ $0+2=2$	$T[0][2] + cost(q_0, q_1, q_2)$ $0+7=7$	$T[0][3] + cost(q_0, q_1, q_2, q_3)$ $0+14=14$
0	$T[1][0] + cost[q_0]$ $0+0=0$	$min \begin{cases} T[1][1] + cost[q_1] \\ T[1][2] + cost[] \end{cases}$ $min \begin{cases} 0+0=0 \\ 0+2=2 \end{cases}$	$min \begin{cases} T[1][1] + cost[q_1, q_2] \\ T[1][2] + cost[q_2] \\ T[1][3] + cost[] \end{cases}$ $min \begin{cases} 0+5=5 \\ 2+0=2 \\ 7+0=7 \end{cases}$	$min \begin{cases} T[1][1] + cost[q_1, q_2, q_3] \\ T[1][2] + cost[q_2, q_3] \\ T[1][3] + cost[q_3] \\ T[1][4] + cost[] \end{cases}$ $min \begin{cases} 0+7=7 \\ 2+2=4 \\ 7+0=7 \\ 14+0=14 \end{cases}$
0	$T[2][0] + cost[q_0]$ $0+0=0$	$min \begin{cases} T[2][1] + cost[q_1] \\ T[2][2] + cost[] \end{cases}$ $min \begin{cases} 0+0=0 \\ 0+0=0 \end{cases}$	$min \begin{cases} T[2][1] + cost[q_1, q_2] \\ T[2][2] + cost[q_2] \\ T[2][3] + cost[] \end{cases}$ $min \begin{cases} 0+5=5 \\ 0+0=0 \\ 2+0=2 \end{cases}$	$min \begin{cases} T[2][1] + cost[q_1, q_2, q_3] \\ T[2][2] + cost[q_2, q_3] \\ T[2][3] + cost[q_3] \\ T[2][4] + cost[] \end{cases}$ $min \begin{cases} 0+7=7 \\ 0+2=2 \\ 2+0=2 \\ 4+0=4 \end{cases}$

of their cluster, hence they are good candidate to place the snapshots for materialization.

Proposition 8 *Given $T_q^* = \{q_0, q_1, ..., q_n\}$ as the query workload on the temporal relation r^T, we would like to group the $q_i \in T_q^*$ into m number of "clusters".*

Applying K-means clustering methodology to this problem requires to minimize the objective function defined as:

$$J = \sum_{j=1}^{m} \sum_{i=1}^{n} ||q_i^{(j)} - \mu_j||^2$$

where μ_j is the centroid of j^{th} cluster and $||q_i^{(j)} - \mu_j||^2$ is the squared error function which indicates the distance between each query and their assigned centroids. Minimizing objective function is achieved by the relocation of μ_j until no changes occur in the objective function.

The K-means clustering algorithm is given as Algorithm 8.

Function `K-Means`($T_q^*\{q_1, ..., q_n\}$, *m*, *maxIteration*)**:**
  iteration $\leftarrow 0$
  repeat
    $\{\mu_1, ..., \mu_m\} \leftarrow SelectRandomSeeds(\{q_i \in T_q^*\}, m)$
    for $i \leftarrow 1$ **to** n **do**
      $J \leftarrow argmin_{J^*} ||\mu_{J^*} - q_i||^2$
      $\mathcal{L}_j \leftarrow \mathcal{L}_j \cup \{q_i\}$
    end
    for $j \leftarrow 1$ **to** m **do**
      $\mu_j \leftarrow \frac{1}{\mathcal{L}_j} \sum_{q \in \mathcal{L}_j} q$
    end
    iteration $++$
  until *convergence and iteration* $\leq$ *maxIteration*
  return $\{\mu_1, ..., \mu_m\}$

Algorithm 8: K-means clustering to compute m number of segmentations

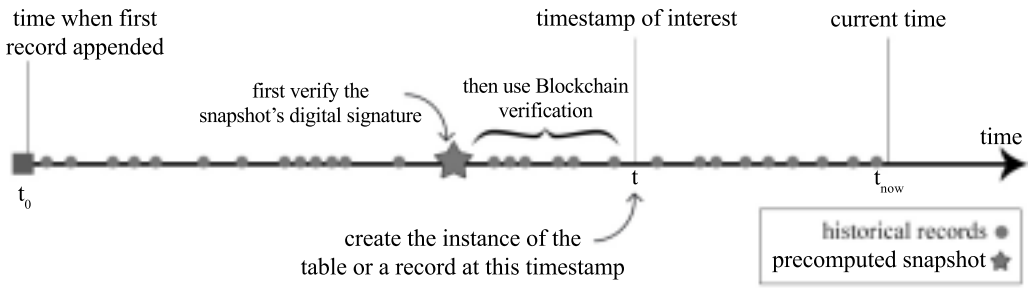

FIGURE 3.10: verify trustworthiness of the records in snapshot materialization

3.5 Trusted Snspshots for materialization

In order to reduce the cost of Blockchain verification and make it inline with snapshot materialization strategy, we suggest the trusted snapshots.

Definition 17 (Trusted Snapshots) *The trusted snapshot s^* is a table with attributes $attr(s) \cup \{signature\}$ where*

$$tail(s^*[signature]) = signature(\sum_{i=0}^{n}(rec_i) : rec_i \in s)$$

Definition 18 (Trusted snaphsot materialization) *To materialize the trusted snapshots for Blockchain verification, the following steps should be taken:*

- ***step 1.*** *the signature of the materialized snapshot to be checked.*
- ***step 2.*** *The trustworthiness of the records which fall in between the query q and snapshot s to be verified using blockchain verification.*

These steps could also be depicted as Figure 3.10.

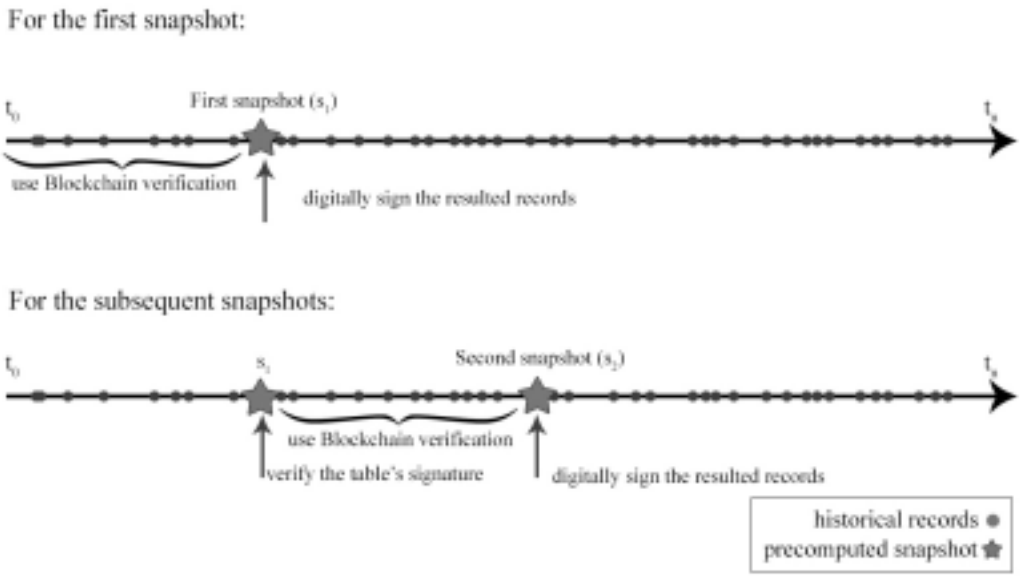

FIGURE 3.11: rules that needs to be followed when signing pre-computed snapshots

The remaining issue is that, to digitally sign the records in s, we need to make sure about the trustworthiness of the records in the snapshot beforehand. Therefore we propose the following rules for that purpose:

- The first snapshot's records trustworthiness is checked using blockchain verification.
- The subsequent snapshots materialize their previous snapshot, hence we take the same steps that was proposed in Definition 18.

Figure 3.11 depicts the rules that needs to be followed when signing a pre-computed snapshot.

3.6 Discussion

Utilizing Blockchain technology to provide verifiablilty and immutability for the records that are stored in a temporal relation discussed in this chapter. We

presented a way to store security information of the transactions also known as proof of work to the temporal relations. this is basically achieved by adding the information of the transaction submitter along with the digital signature of the transaction using submitter's cryptogrphic keys. Digital signature of the transactions enables all the users who have access to the public key of the submitters to verify whether or not the submitted records are authenticate or not.

Although digital signatures grant verifiability of the records but the tables with the security information are still vulnerable to malicious attacks. For example, a super user of the database who can bypass all the security requirements can still perform unverifiable malicious attacks. Therefore, it is naive to assume that using merely the digital signatures can guarantee the trustworthiness of the records.

To solve the issue, we proposed chaining the submitted records together using their digital signature. This could be done by using the Blockchain technology in which the data are represented in the form of blocks and each block contains the digital signature of their previously submitted block. Any changes on the blocks result in a completely different signature that brings inconsistancy in the chain of digital signatures. To implement this idea, we proposed to add the *previousSignature* attribute to the records in the temporal table that contains the digital signature of their previous record.

In order to verify the chain of digital signatures, each record in the temporal relation need to be visited and the correctness of their digital signature as well as the correctness of their previous record's digital signature needs to be investigated. We call this the manual blockchain verification method which is a linear pocess.

Also, given a temporal relation which contains historical data of the database, for the sake of decision making and data analytics, it is common to query for

the version of the relation in an specific timestamp. These queries could be answered by creating the snapshot of the relation until that timestamp. The process of snapshot creation is also linear as all the records that fall before the query timestamp needs to visited once and the updates on them needs to be applied.

In this chapter, We reasoned that answering to snapshot creation queries and manual blockchain verification is expensive and inefficient, therefore we proposed to place m number of snapshots for materialization. These snapshots indicate the latest version of a database until that specific timestamp. Note that m which is the number of snapshots to be generated and placed for materialization is determined by checking the available resources of the host system.

To find optimal locations for placing snapshots, we proposed to store the timestamp of the previous queries on the database to have the patterns of the queries on the timeline. Pattern of the queries gives us the ability to identify the hotspots on the timeline. We argued that it is highly likeley that the the subsequent queries to be performed in these hotspots. in order to solve the problem of finding optimal locations for the snapshots, we proposed to create optimal segmentations of the previously performed queries and allocate an exclusive snapshot for each segmentation for materialization. In this case, when a query falls in the boundaries of a segmentation, it materializes the snapshot of that segmentation.

In order to find m number of optimal segmentation, we first solved the problem of finding an optimal position to place a single snapshot. We mathematically proved that the optimal place for a single snapshot is the median of the queries, hence if m number of segmentations were created, the optimal position to place snapshot within each segmentation is the median of that segmentation queries. In the next step, we utilized three different methods to find multiple optimal segmentation of queries: recursive algorithm, dynamic programming

and heuristic method. In theory, the recursive algorithm consumes the highest computational time among the other two methods. Also, computationally, heuristic method has the lowest computational time complexity over recursive algorithm and dynamic programming, but the optimal solution to the problem is not always guaranteed. In the next chapter we design multiple experiments to evaluate the performance of each method in different scenarios.

With the intention of providing trusted query results, we proposed to create the digital signatures of the snapshots and append it to the end of the snapshot table. Digitally signing the precomputed snapshots requires to investigate the authenticity of their records before signing them. This process requires manual blockchain verification which is costly, especially for the snapshots that fall at the end of the timeline. In order to save the computational time, we proposed that only the first snapshot use the manual Blockchain verification and the subsequent snapshots use the materialized Blockchain verification using their previous precomputed snapshot.

Chapter 4

System Implementation And Experimental Evaluation

This chapter attempts to discusses the experiments performed to evaluate the proposed solutions for the implementation of the system. Section 4.1 provides information about the system design and choice of software and programming languages. Section 4.2 provides an evaluation of discussed methods for snapshot materialization by setting up different experiments.

4.1 Development environment and application development tools

To setup the development environment of the project, we used the 'Leda' research server provided by the department of scinece at the University of Ontario Institute of Technology with the ubuntu 16.04 operating system installed. The application could be developed by various tools, but since one important goal of this project was to be able to generalize our proposed solution for majority of relational databases, we needed to choose the tools that are popular among the developers for the development of their system. For the database choice,

we favored the use of postgreSQL simply because of its wide adoptation by the industry [4].

The experiments scripts were mainly developed by using Python 2.7 programming language. We started off by generating a database using the TPCH schema with 1,000,000 tuples in the main table. Using Python's pycrypto 2.6.1 library [1], we developed a few functions to create digital signature of the transactions on the database system. We generated a synthetic temporal database with security information, by performing a set of random insert, delete and update transactions and by creating their digital signature at 1000 distinct timestamps. In each record, along with the digital signature of that record, we also stored the digital signature of its previous record in the temporal table. This created a Blockchain-based relational temporal table with 1,000 timestamps.

To be able to manage the relational database and perform queries on the tables, we utilized SQL query language. Generation of snapshots using the records stored in the temporal tables required time series analysis and performing numbers of aggregations and self-joins on the relational table. In order to make this process less cumbersome, we utilized the windowing function that is part of the SQL standard and is supported by the relational databases [20].

In order to evaluate the effectiveness of snapshot materialization, the optimal snapshot placement had to be examined in different query distributions on the timeline. The Python's Numpy library gave us the ability to simulate some query distributions on the timeline. The recursion and dynamic programming experiments was performed using Python 2.7 programming language. The heuristic method experiment, also is based on scikit-learn machine learning library [2] for Python. After running the snapshot materialization experiments

[1] https://pypi.org/project/pycrypto/
[2] http://scikit-learn.org/stable/

and collecting data, we utilized the Python's plotly library in order to visualize the collected data.

4.2 Experimental evaluation

In this section we discuss the evaluation of snapshot materialization through different experiments. We start by showing a method to create snapshots using the temporal relation. Then we show the problem of linear computational time in performing snapshot generation on the temporal relation. Next, we evaluate the placement of a single snapshot for matrialization in different timestamps and visualize the optimal position for this purpose. In the next step we show the effectiveness of utilizing multiple number of precomputed snapshots for materialization and at the end, multiple approaches to find multiple optimal timestamps are put into different experiments.

4.2.1 Generating snapshots by using records in a temporal database

We can construct the snapshots using simple windowing functions (as is supported by PostgreSQL [23]). The pseudo code of the developed function is shown in Table 4.1.

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.

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

TABLE 4.1: The windowing function to create snapshots

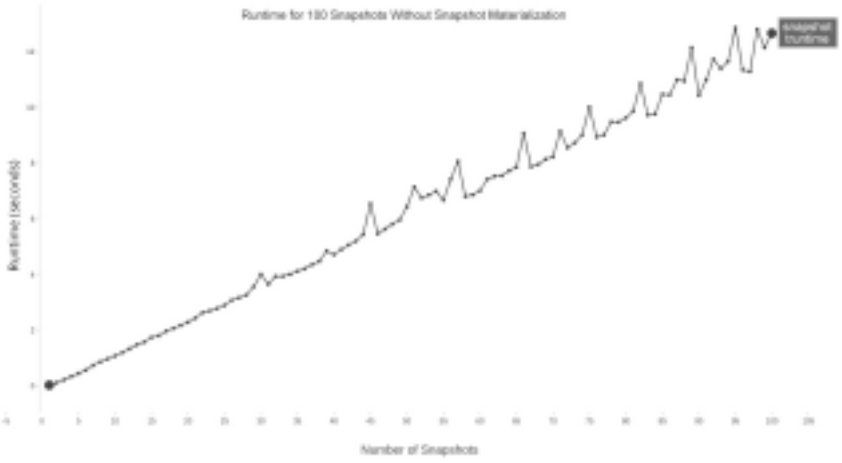

FIGURE 4.1: Linear time in computation of snapshots without snapshot materialization

4.2.2 Evaluating the linearity of snapshot generation

To prove the linearity of snapshot generation experimentally, we developed an experiment to create and record runtime of 100 snapshots on the timeline of temporal database. Using the resulted runtime data, the cost-line was visualized which is shown if Figure 4.1. Note that the runtime is in seconds.

Figure 4.1 clearly proves our claim that computation of snapshots on a temporal table requires linear time. The issue of linearity in computation of snapshots makes such tasks on a very large table inefficient.

4.2.3 Evaluating materialization of a single snapshot

The purpose of this experiment is to visually see that the optimal timestamp of a single snapshot for materialization is the median of previousely performed queries. For this purpose we randomly sampled 60 query timestamps from our temporal database timeline. Each query was genrating an snapshot of the relation at an specific timestamp. In the next step we placed a single snapshot for these queries to materialize and computed the overal cost of query answering using definition 15. To see the effects of repositioning the materialized snapshot on the timeline, we slided the materialized snapshot on the timeline of the temporal relation. The resulted overall cost-line obtained from sliding the snapshot is depicted in Figure 4.2.

The resulted cost line shown in Figure 4.2 clearly shows that the position of a snapshot directly affects the overal cost of query answering on the temporal table. We also computed the median of the queries on the timeline which is seen as a red dot on the figure. As seen in the figure, the median of the queries is indeed located in the position where the cost-line's global minimum is. This proves that the median of the queries is the most optimal position to place a single snapshot for materialization.

4.2.4 Evaluating materialization of multiple snapshot

To evaluate the performance of our optimal snapshot computation, we evaluated the recursive formulation given by Section 3.4.1, the dynamic programming formulation given by Section 3.4.2 and heuristic method given by Section 3.4.3.

To illustrate that the optimal snapshot placement indeed produces the best query answering performance, we compared the query answering cost of three approaches:

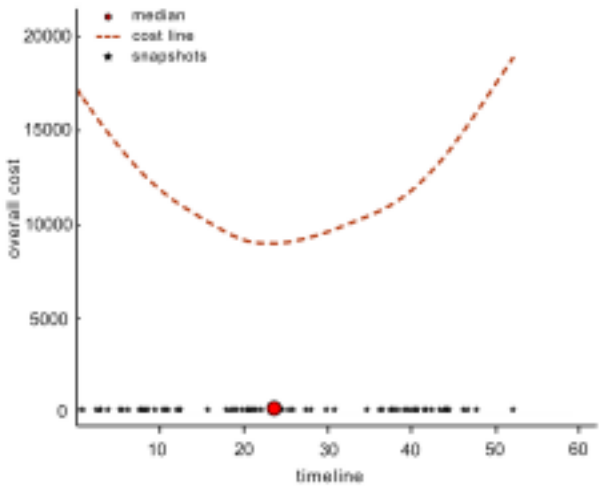

FIGURE 4.2: Cost of query answering using a single snapshot over different snaptshot timestamps

- Pick m random timestamps to place the snapshots.
- Pick m evenly intervaled timestamps to place the snapshots.
- Pick m timestamps computed by dynamic programming.

Figure 4.3 shows that the placements obtained by dynamic programming clearly beats the other two approaches.

In order to evaluate the effectiveness of m number of snapshots to lower the overal cost of query answering, we recorded the overal cost of queries in various number of snapshots. Figure 4.4 shows that as the number of snapshots for materialization increases, the overal cost of answering to the queries drops.

4.2.5 Evaluating approaches for optimal placement of m snapshots

In this research, we examine three approaches to find the optimal timestamps for m number of snashots for materialization. The performed approaches are:

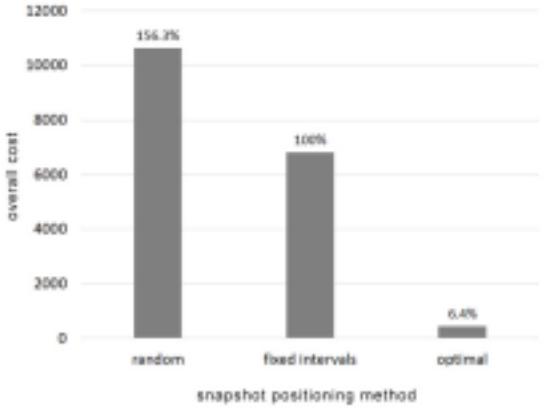

FIGURE 4.3: Query answering cost with forty snapshots with various approaches to place snapshots for materialization

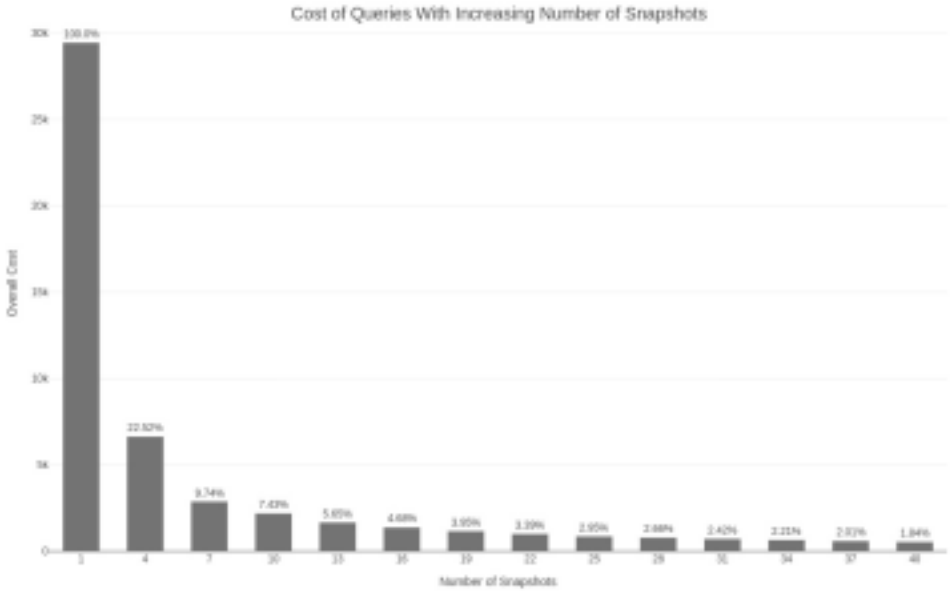

FIGURE 4.4: Query answering cost with increasing number of snapshots

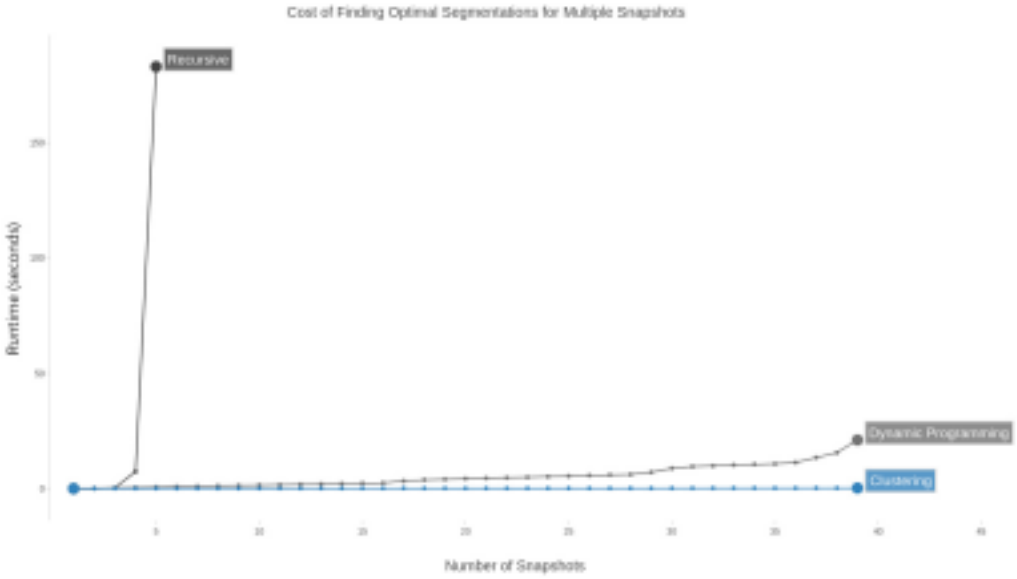

FIGURE 4.5: Optimization runtime with respect to the number of snapshots (in seconds)

- Recursive algorithm.
- Dynamic programming.
- K-Means clustering.

Experiment 1.

In the first experiment , we evaluate the performance of each approach with respect to fixed number of queries and variable number of snapshots For this experiment we sampled 140 number of queries and evalueted the runtime of each approach while increasing the number of requested snapshots.

Figure 4.5 and Table 4.2 depict the observations of the experiment. The graph shows that in comparison with dynamic programming and K-means clustering, the recursive algorithm is computationally more expensive. In fact we could not

TABLE 4.2: Optimization runtime with respect to the number of snapshots

Snapshots	Recursive (sec)	Dynamic (sec)	Clustering (sec)
1	0.0002	0.01	0.01
2	0.01	0.17	0.02
3	0.32	0.34	0.03
4	7.39	0.52	0.04
5	183.18	0.69	0.06
10	N/A	1.52	0.08
15	N/A	2.33	0.12
20	N/A	4.33	0.12
25	N/A	5.46	0.14
30	N/A	8.81	0.17
35	N/A	10.74	0.21
40	N/A	21.09	0.24

find the optimal timestamp for more than five snapshots because the results did not converge.

Figure 4.5 and Table 4.3, have a closer look at the dynamic programming and clustering method. As it could be seen, the clustering technique finds the optimal timestamps for the snapshots in a lower runtime.

Exmperiment 2.

In the second experiment of this category, we evaluated the three approaches of finding optimal timestamps for fixed the number of snapshots and variable number of queries. For this reason, we computed 4 number of optimal timestamps for queries ranging from 12 to approximately 180.

Figure 4.8 and Table 4.5 shows that similar to the previous experiment, recursive algorithm has proven to be expensive in computing the optimal timestamps

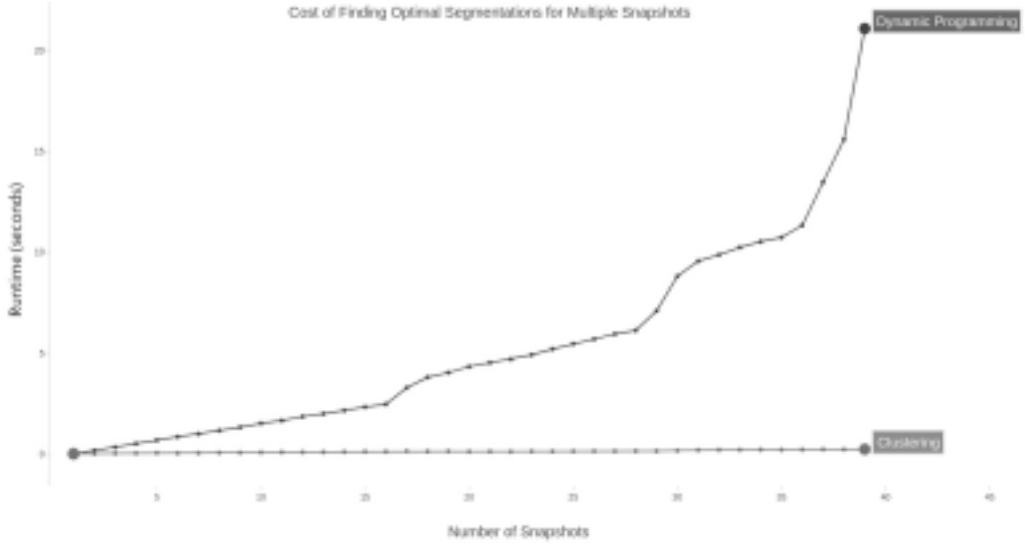

FIGURE 4.6: Optimization runtime with respect to the number of snapshots (in seconds)

TABLE 4.3: Optimization runtime with respect to the number of snapshots

Snapshots	Dynamic (sec)	Clustering (sec)
1	0.01	0.01
2	0.17	0.02
3	0.34	0.03
4	0.52	0.04
5	0.69	0.06
10	1.52	0.08
15	2.33	0.12
20	4.33	0.12
25	5.46	0.14
30	8.81	0.17
35	10.74	0.21
40	21.09	0.24

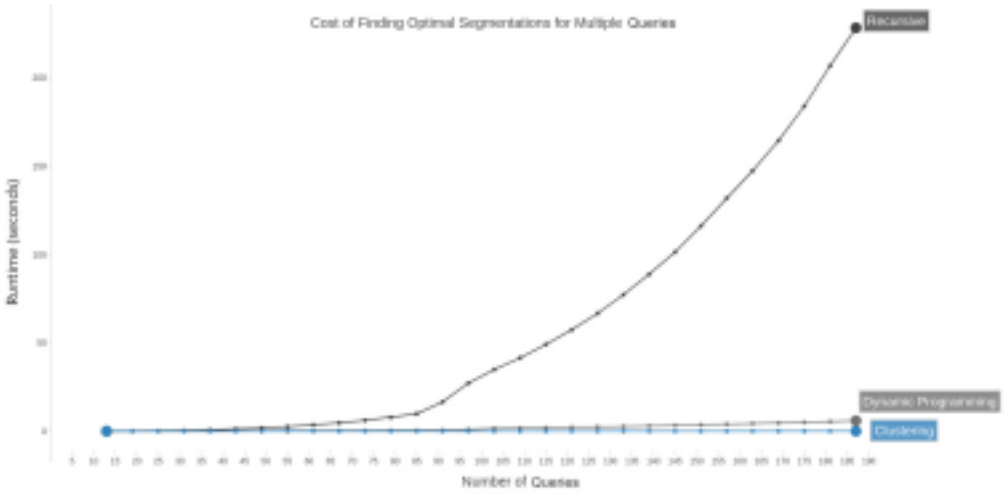

FIGURE 4.7: Optimization runtime with respect to the number of queries (in seconds)

TABLE 4.4: Optimization runtime with respect to the number of queries

Queries	Recursive (sec)	Dynamic (sec)	Clustering (sec)
13	0.05	0.01	0.02
43	1.23	0.13	0.03
73	6.15	0.39	0.03
103	34.98	1.60	0.03
133	77.31	0.69	0.04
163	147.57	4.35	0.04
187	228.52	5.96	0.04

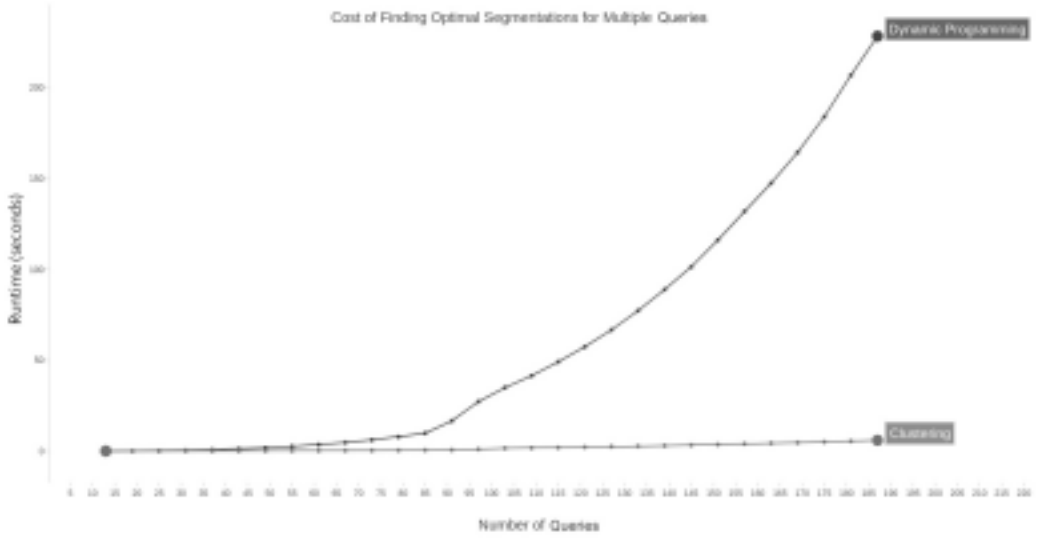

FIGURE 4.8: Optimization runtime with respect to the number of queries (in seconds)

TABLE 4.5: Optimization runtime with respect to the number of queries (in seconds)

Queries	Dynamic (sec)	Clustering (sec)
13	0.01	0.02
43	0.13	0.03
73	0.39	0.03
103	1.60	0.03
133	0.69	0.04
163	4.35	0.04
187	5.96	0.04

of the snapshots. Figure 4.8 and Table 4.5 also compares the dynamic programming with heuristic method.

The results obtained from these set of experiments show that, the heuristic method is computationally more favorable than the other two methods. When in a system, the *exact optimal* solution has more priority over the runtime, the dynamic programming could be seen as a better option. However in largescale systems that the runtime is more important, heuristic method could be more

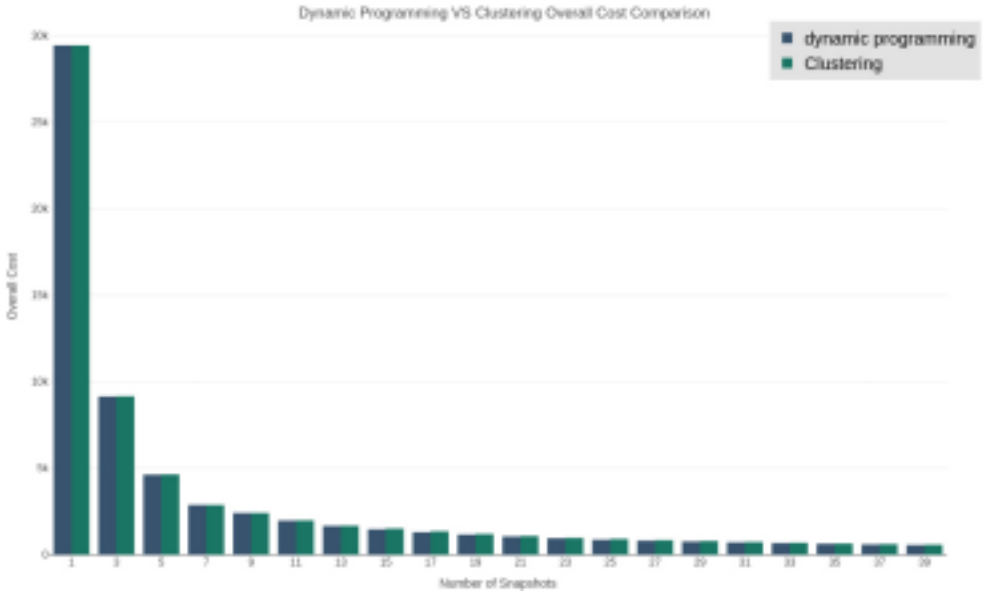

FIGURE 4.9: Comparing the outcome of dynamic programming with the heuristic method

favorable. Mobile systems are one example in which the runtime matters the most.

4.2.6 Evaluating the heuristic method

Since heuristic method does not guarantee the *exact* optimal solution, we tend to compare the solution obtained from heuristic method with the *exact* optimal solution obtained from dynamic programming method. Our objective was to see if the heuristic method returns satisfactory results. In this experiment, we compared the outcome of the dynamic programming and K-Means clustering method in varying number of snapshots but fixed number of queries (160 queries). Figure 4.9 and Table 4.6 show that the difference from outcome of dynamic programming and heuristic method is slight, and the heuristic method has a satisfactory results.

TABLE 4.6: Comparing the outcome of dynamic programming with the heuristic method

Snapshots	Dynamic	Clustering
1	29439.26	29439.26
3	9141.55	9159.13
5	4626.77	4630.08
7	2867.74	2867.74
9	2410.46	2412.62
11	1972.14	1980.95
13	1664.14	1673.32
15	1471.58	1509.57
17	1300.32	1351.44
19	1162.25	1194.61
21	1051.97	1079.71
23	951.06	970.72
25	867.35	907.30
27	810.01	836.97
29	759.69	787.86
31	713.04	731.61
33	670.39	684.13
35	629.69	640.58
37	591.08	608.96
39	557.81	575.97

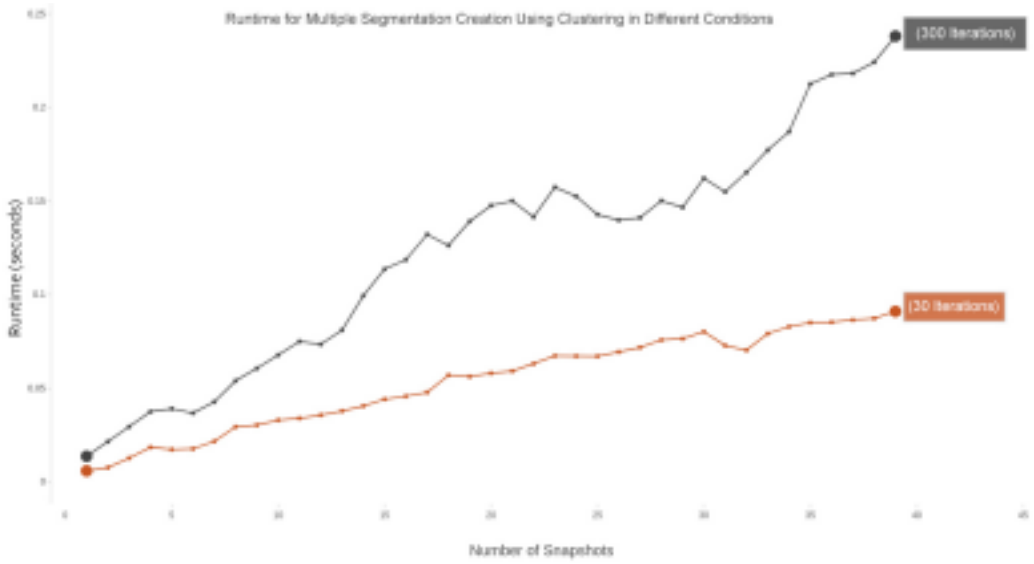

FIGURE 4.10: Comparing the runtime of K-Means clustering method with 30 and 300 iterations for variable number of snapshots

The K-means clustering method used in the experiments used 300 itterations until the solution is converged. For the purpose of lowering the runtime even more, we lowered the number of iterations to 30 and performed the experiments again. Figure 4.11 shows the runtime comparison of K-Means clustering method with 300 iterations and 30 iterations for fixed number of queries but variable number of snapshots and Figure 4.10 shows the same comparison for fixed number of snapshots but varying number of queries.

Then we compared the overal cost of query answering with the same setting in variable number of snapshots. As it could be inferred from Figure 4.12 and Table 4.7, loweing the number of iterations lowers the precision of finding an optimal solution but the results are still satisfactory.

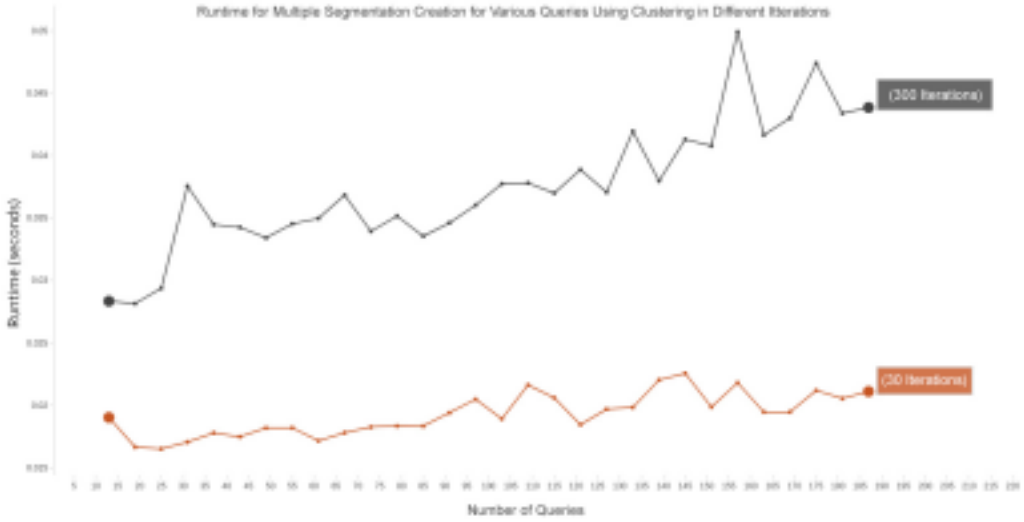

FIGURE 4.11: Comparing the runtime of K-Means clustering method with 30 and 300 iterations for variable number of queries

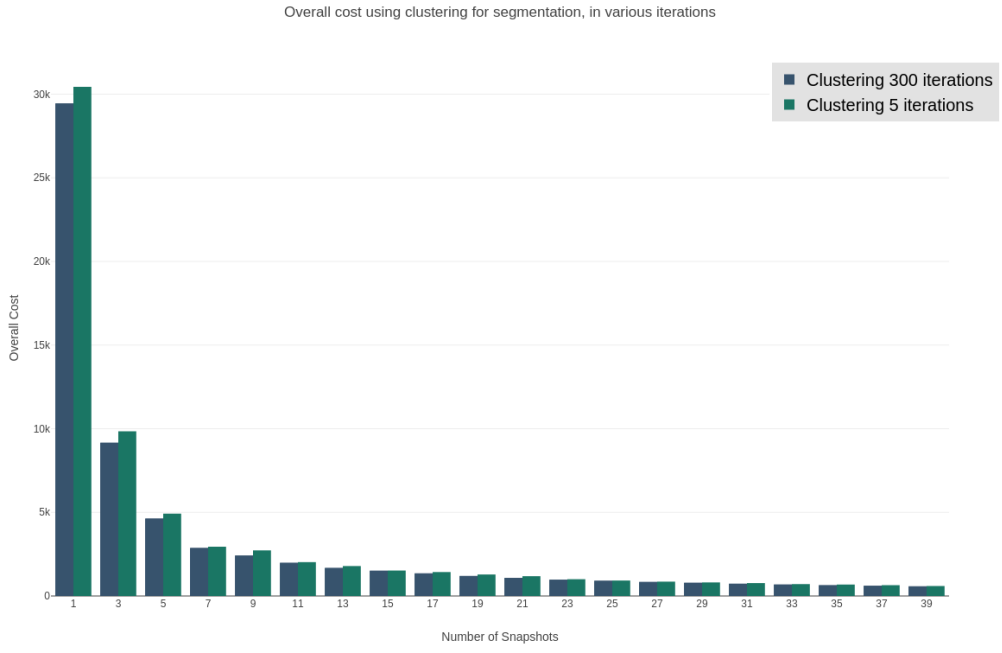

FIGURE 4.12: Comparing the overal cost of query answering in variable number of snapshots using K-Means clustering method with 30 and 300 iterations to find optimal timestamps for snapshots

TABLE 4.7: A comparison between K-means clustering with 30 and 300 iterations

Snapshots	300 iterations	5 iterations
1	29439.26	30439.26
3	9159.13	9841.13
5	4630.08	4921.08
7	2867.74	2945.76
9	2412.62	2732.38
11	1980.95	2023.24
13	1673.32	1789.50
15	1509.57	1521.68
17	1351.44	1430.66
19	1194.61	1284.61
21	1079.71	1182.71
23	970.72	1002.57
25	907.30	923.30
27	836.97	856.83
29	787.86	811.08
31	731.61	769.83
33	684.13	710.49
35	640.58	683.60
37	608.96	645.74
39	575.97	595.89

4.3 Discussion

In this chapter the effectiveness of the proposed solutions to both lower the cost of queries on the temporal tables and add verifiability to the stored records in a database were put into experiment. We argued that the system could be developed by various tools but in order to generalize the application of our proposed methods for majority of relational databases, we chose the most favorable programming languages and tools by developers. The effectiveness of the snapshot materialization was the first set of experiments which was carried out. The experiments proved our claim that for a single snapshot placement for materialization, the median of the perviously performed queries is the most optimal timestamp. The experiments also showed that for calculating the multiple optimal segmentation of queries on the timeline, the heuristic method beats the recursive and dynamic programming method in terms of computational time complexity. Our experiments showed that although the optimal solution is not guaranteed in the heuristic method, but the results obtained by using this method are satisfactory. For the sake of reducing the computational time even more, we lowered the iteration of the heuristic method and the results obtained were still satisfactory.

Briefly speaking, the advantages and disadvantages of utilizing recursive algorithm, dynamic programming and heuristic method could be shown in Table 4.8.

TABLE 4.8: Comparison between the methods to create optimal segmentations

method	Pros	Cons
Recursive algorithm	Exact optimal solution	Expensive for large number of queries and segmentations
Dynamic programming	Exact optimal solution	Expensive for large number of queries and segmentations
	Cheaper than recursive algorithm	
K-Means clustering	Fast in computation	optimal solution not guaranteed

Chapter 5

Conclusion

5.1 Summary

In this work, implementation of a Blockchain based temporal database was proposed that works based on Relational Database Management System (RDBMS). The trustworthiness of the records in the temporal relations is verifiable using immutable digital signatures. The temporal relation contains all the updates of a particular relation, therefore it could be used as the source of data provenance for that relation. To implement the system, two major problems were identified:

- Make the historical records stored in the temporal table immutable and verifiable.
- Lower the cost of snapshot generation on the temporal database.

Using Blockchain to add immutability and verifiability for the records of temporal relations, require these relations to have a few security information attributes: the transaction submitters information, the digital signature of the record signed by transaction submitter's cryptoraphic keys and the previous record's digital signature. Creating the digital signature of the records and storing them with the record provides verifiability for that records. Also storing the

digital signature of the previous record in current record, creates the chain of records also known as Blockchain of records. The Blockchain which is created by doing this, ensures that the malicious attempts on the temporal database, result in an evident inconsistent data. The validation of the chain of records is done by

- validate the authenticity of each record by verifying their digital signatures.
- making sure that the current record's *'previous signature' field, matches the previous record's 'current signature' field for all the records in the table.*

The process of verifying the chain is said to be linear because there is a need to visit every single record of the temporal relation and verify their trustworthiness.

Having a temporal database as the source of data provenance, enables users to create snapshot of the relation in an specific timestamp. In fact for the purpose of decision making and data analytics such queries are common. In this thesis, the problem of linear time and storage to perform such queries on the temporal databases shown both theoretically and experimentally. In the presence of multiple and concurrent queries on the temporal table, performing snapshot creation queries are inefficient and infeasable.

In oder to reduce the cost of snapshot creation queries, the materialization of multiple precomputed snapshots proposed. The proposal advices that keeping the timestamp of previous queries which was performed on the temporal table gives insight into the pattern of queries on the table and identifies hotspots. These hotspots are useful because there is a high chance that the subsequent

queries to be performed in those hotspots. Optimal segmentation of the performed queries, creates optimal clusters of the queries and specifies a precomputed snapshot for each cluster to materialize. It is expected that this method result in lowering the overall cost of query answering on the temporal database.

For the placement of a single snapshot for materialization, both mathematical and experimental approach showed that the most optimal timestamps is the median of previousely performed queries. This notion could be extended for placing multiple precomputed snapshots for materialization. That is, when optimal multiple segmentations of the queries were computed, the optimal place in each segmentation for placing the snapshot is the median of queries in that segment.

Finding the optimal multiple segmentations of the previousely performed queries was performed using three different aproaches: recursive algorithm, dynamic programming and heuristic method. The experiments showed that the heuristic method does not guarantee the most optimal solution, but it is more favorable than the other two methods because of less computational time and satisfactory results.

To extend the idea of snapshot materialization for the problem of Blockchain verification, it is recommended to create a digital signature of the precomputed snapshots, and place it to the end of each snapshot table. By doing so, the validation of the Blockchain, when a query is performed is done by:

- check the digital signature of the snapshot that the query is materializing.
- for the records that fall in between the query timestamp and snapshot timestamp, use the manual chain validation technique

5.2 Conclusion

In this thesis, the development of a trusted relational temporal database was discussed. The temporal database contains the historical data of a database which makes them a suitable source of data provenance for the database system. In order for the temporal database to be trusted, the records in it must be immutable and verifiable. For this purpose, digital signature is a suitable tool to be utilized. Digital signatures provide verifiability for the stored records and also make undetectable forgeires on the temporal database difficult. Using digital signatures in the context of this research is achieved by creating a signature of the submitted transaction using the submitter's private key. Moreover, To reduce the risk of adverserial attacks, especially from the super-users of the system, the records that are digitally signed could be chained together by a record holding its previous record's digital signature. By doing so, any malicious or accidental data manipulation on the temporal relation, results in an inconsistent data.

The temporal databases that we talked about can become massive as they store all the updates on the relations of a database. This makes querying for the snapshot of a relation on them expensive as there is a need to compute a large workload for each query. Snapshot materialization is a suitable way to reduce the cost of query answering on the temporal databases. For this purpose, multiple number of snapshots could be precomputed for materialization. Finding the optimal timestamps to place the snapshots could be started by storing the timstamp of performed queries on the temporal relation. By doing so, a useful information about the hotspots of the timeline of temporal table is provided. If one snapshot is to be placed, the median of queries is the most optimal place for it. For multiple snapshots, optimally creating segmentations of the queries and placing snapshots in the median of each segmentation, might an optimal

solution.

Creating optimal segmentation could be seen as an optimization problem and could be computed using recursive algorithm, dynamic programming and heuristic method. Experiments proved that in this project, the heuristic method is more favorable because it provides satisfactory results in a much affordable runtime. Multiple experiments also proved our claim that the optimal snapshot materialization can reduce the cost of query answering more significant than placing materialized snapshots randomly or in fixed intervals.

At last, to guarantee that the snapshots created using materialized view is trustworthy, digital signature of the materialized snapshot could be created and added to their table. This is a beneficial method that reduces the cost of Blockchain verification by materializing the snapshots.

5.3 Future work and other remarks

Many different extensions and changes to the proposed system has been left for the future work due to the complicateness of the problems and the limitation of time. Future works contain different possible applications of the proposed system and methods which can make the current system perform better. It would be interesting to consider the following applications, extensions and methods of the proposed system.

5.3.1 Utilization in distributed storages on P2P networks

In recent years decentralized file systems have become a trend in business environments. Unlike centralized storages in which all data of a system stored on central servers and any data access should be authorized by these central

servers, in decentralized file systems, computers can share information with each other directly without having an intermediary involved [18]. Current decentralized file systems such as *Interplanetrory File System* (IPFS), *Storj* and *Sia* are distributed storages on peer to peer networks that do not rely on any central service provider to monitor and permit file sharing between the nodes [40].

Many of decentralized file systems utilize Blockchain technology in order to make digital assets that are shared between the nodes secure and virtually immutable [40]. For example, IPFS utilizes *Filecoin* which is basically a Blockchain that is built on top of IPFS and guarantees secure file replication and synchronization between the nodes [36].

It would be interesting to employ the proposed trusted temporal relation to not only provide data provenance for the records in such decentralized file systems but also make queries on the relation as well as verifying the authenticity of the Blockchain computationally cheaper. However, in order to achieve the objective there are multiple problems that need to be solved.

Finding the optimal snapshot materialization strategy in the distributed file system is the most flagrant problem. In this thesis we argued that the optimal snapshots could be placed in the center of hotspots on the timeline of the temporal database. We discussed that these hotspots could be identified by capturing the timestamp of past queries. However the usefulness of this strategy for a decentralized network needs to be investigated. It needs to be inquired if it is optimal to either place snapshots for every individual node on the decentralized network or it should be calculated once for all the nodes. The other possible question is where to store these snapshots on the network. All mentioned and many other problems create a new dimension to the optimal snapshot materialization problem.

5.3.2 Trust certification

In the world wide web environment, in order to provide secure connections between two parties in the network, *Secure Socket Layer* (SSL) certificates are used that encrypts all the interactions between a browser and a web server. Similar to that, based on our proposed system, a user interface with certification could be created that reflects the "*Trust*" in the data exploration. This cerificate guarantees that the stored data of a database and any results from queries on them are trustworthy.

5.3.3 More efficient clustering methods

Another evaluation which is absent in this study is the comparison between K-Means clustering method with other clustering techniques. Other clustering techniques might serve as heuristics with more precise results or lower computational complexity.

5.3.4 Generalization for different database models

This system was designed based on the Relational Database Management System (RDBMS) which makes it inapplicable for other database models. For example, graph databases who has gained popularity as an alternative to the relational model, has different method of storing and retrieving data than the relational databases. In order to generalize the implementation of the system for different database models, further studies need to be conducted.

5.4 Limitations

Our methodology has a number of limitations which are discussed in this section. In this research, one of the steps to provide trust over the historical records is to make them immutable. This requires that the temporal relation's attributes to be immutable as well. This limits the users to add or remove any attributes of the temporal relation.

Another limitation is that the optimization method discussed in this research is static. When the query workload change over time, the optimal location of snapshots also change. With the current proposed method, instead of dynamically computing the optimal locations, the optimal location must be recomputed again.

Bibliography

[1] Jean-Philippe Aumasson et al. *The Hash Function BLAKE*. Berlin, Heidelberg: Springer, 2014.

[2] Amin Beirami, Ken Pu, and Ying Zhu. "Towards Optimal Snapshot Materialization To Support Large Query Workload For Append-only Temporal Databases". In: *2018 IEEE BigData Congress*. 2018, pp. 268–271.

[3] James Cheney, Laura Chiticariu, and Wang-Chiew Tan. *Provenance in Databases: Why, How, and Where*. Now Pub, 2009.

[4] Joshua Cook. *Docker for Data Science Building Scalable and Extensible Data Infrastructure Around the Jupyter Notebook Server*. Apress, 2017.

[5] Scott A. Crosby and Dan S. Wallach. "Efficient data structures for tamper-evident logging". In: *2009 the 18th conference on USENIX security symposium*. USENIX Association. 2009, pp. 317–334.

[6] Chenyun Dai et al. "An Approach to Evaluate Data Trustworthiness Based on Data Provenance". In: *Secure Data Management*. Springer Berlin Heidelberg, 2008, pp. 82–98.

[7] *Department of Defense Trusted Computer System Evaluation Criteria*. Standard. United States Department of Defense, 1985.

[8] Vikram Dhillon, David Metcalf, and Max Hooper. *Blockchain Enabled Applications*. Apress, Berkeley, CA, 2017.

[9] Jiang Du et al. “DeepSea: Progressive Workload-Aware Partitioning of Materialized Views in Scalable Data Analytics.” In: *EDBT*. 2017, pp. 198–209.

[10] Rahul Dutta and Annappa B. “Privacy and trust in cloud database using threshold-based secret sharing”. In: *2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. 2013, pp. 800–805.

[11] Daniel Fabbri, Ravi Ramamurthy, and Raghav Kaushik. “SELECT triggers for data auditing”. In: *2013 IEEE 29th International Conference on Data Engineering (ICDE)*. 2013, pp. 1141–1152.

[12] Devarshi Ghoshal and Beth Plale. “Provenance from Log Files: A BigData Problem”. In: *Proceedings of the Joint EDBT/ICDT 2013 Workshops*. EDBT ’13. ACM, 2013, pp. 290–297.

[13] Stuart Haber and W. Scott Stornetta. “How to Time-Stamp a Digital Document”. In: *Advances in Cryptology-CRYPT0’ 90*. Springer Berlin Heidelberg, 1991, pp. 437–455.

[14] Werner K. Hauger and Martin S. Olivier. “The role of triggers in database forensics”. In: *2014 Information Security for South Africa*. 2014, pp. 1–7.

[15] Richard Johnsonbaugh and Marcus Schaefer. *Algorithms*. Pearson, 2003.

[16] Bhushan Kapoor, Pramod Pandya, and Joseph S. Sherif. “Cryptography. A security pillar of privacy, integrity and authenticity of data communication”. In: *Kybernetes* 40 (2011), pp. 1422–1439.

[17] Grace Khayat and Hoda Maalouf. “Trust in real-time distributed database systems”. In: *2017 8th International Conference on Information Technology (ICIT)*. 2017, pp. 572–579.

[18] Lance Koonce. "The wild, distributed world: get ready for radical infrastructure changes, from blockchains to the interplanetary file system to the Internet of things". In: *Intellectual Property and Technology Law Journal* 28.10 (2016), pp. 3–5.

[19] Donald Kossmann and Konrad Stocker. "Iterative dynamic programming: a new class of query optimization algorithms". In: *ACM Transactions on Database Systems (TODS)* 25.1 (2000), pp. 43–82.

[20] Viktor Leis et al. "Efficient Processing of Window Functions in Analytical SQL Queries". In: *Proceedings of the VLDB Endowmen* 8.10 (June 2015), pp. 1058–1069.

[21] Chung-Yi Lin et al. "Secure logging framework integrating with cloud database". In: *2015 International Carnahan Conference on Security Technology (ICCST)*. 2015, pp. 13–17.

[22] Peng Liu, Paul Ammann, and Sushil Jajodia. "Rewriting Histories: Recovering from Malicious Transactions". In: (2000), pp. 7–40.

[23] Bruce Momjian. *PostgreSQL: introduction and concepts*. Vol. 192. Addison-Wesley New York, 2001.

[24] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. https://bitcoin.org/bitcoin.pdf. 2008.

[25] OECD. *OECD Science, Technology and Innovation Outlook 2016*. OECD, 2016.

[26] Jon M. Peha. "Electronic commerce with verifiable audit trails". In: *In Proceedings of ISOC*. 1999.

[27] Doug Rose. *Data Science*. Apress Berkeley,CA, 2016.

[28] Bruce Schneier and John Kelsey. "Cryptographic Support for Secure Logs on Untrusted Machines". In: *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7*. SSYM'98. USENIX Press, 1998, pp. 53–62.

[29] Bruce Schneier and John Kelsey. "Minimizing bandwidth for remote access to cryptographically protected audit logs." In: *In Web Proceedings of the 2nd International Workshop on Recent Advances in Intrusion Detection.* USENIX Press, 1999.

[30] Bruce Schneier and John Kelsey. "Secure Audit Logs to Support Computer Forensics". In: *ACM Trans. Inf. Syst. Secur.* 2.2 (1999), pp. 159–176.

[31] Arunesh Sinha et al. "Continuous Tamper-Proof Logging Using TPM 2.0". In: *International Conference on Trust and Trustworthy Computing*. Springer. 2014, pp. 19–36.

[32] Richard T. Snodgrass, Shilong Stanley Yao, and Christian Collberg. "Tamper Detection in Audit Logs". In: *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*. VLDB '04. VLDB Endowment, 2004, pp. 504–515.

[33] Mohammad Karim Sohrabi and Vahid Ghods. "Materialized View Selection for a Data Warehouse Using Frequent Itemset Mining." In: *Jcp* 11.2 (2016), pp. 140–148.

[34] William Stallings. *Cryptography and Network Security: Principles and Practice (7th Edition)*. Pearson, 2017.

[35] Domenico Talia, Paolo Trunfio, and Fabrizio Marozzo. *Data Analysis in the Cloud: Models, Techniques and Applications*. Elsevier, 2015.

[36] Vatsalya et al. “MARINE HULL INSURANCE USING PRIVATE BLOCKCHAIN, FILECOIN PROTOCOL AND SMART CONTRACTS”. In: *International Journal of Advanced Research in Computer Science* 9 (3 2018), pp. 94–99.

[37] Stratis D Viglas. “Data Provenance and Trust”. In: *Data Science Journal* 12 (2013), GRDI58–GRDI64.

[38] James Wagner et al. “Carving database storage to detect and trace security breaches”. In: *Proceedings of the Seventeenth Annual DFRWS USA*. Elsevier. 2017, s127–s136.

[39] James Wagner et al. “Detecting Database File Tampering through Page Carving”. In: *2018 21st International Conference on Extending Database Technology*. 2018, pp. 121–132.

[40] Shangping Wang, Yinglong Zhang, and Yaling Zhang. “A Blockchain-Based Framework for Data Sharing With Fine-Grained Access Control in Decentralized Storage Systems”. In: *IEEE Access* 6 (2018), pp. 38437–38450.