Towards Trusted Temporal Databases Using Blockchain

By Amin Beirami
MSc student, UOIT

Supervisors: Dr. Ken Pu and Dr. Ying Zhu

October 2018

Trusting data

(Big) Data supports important decision making.

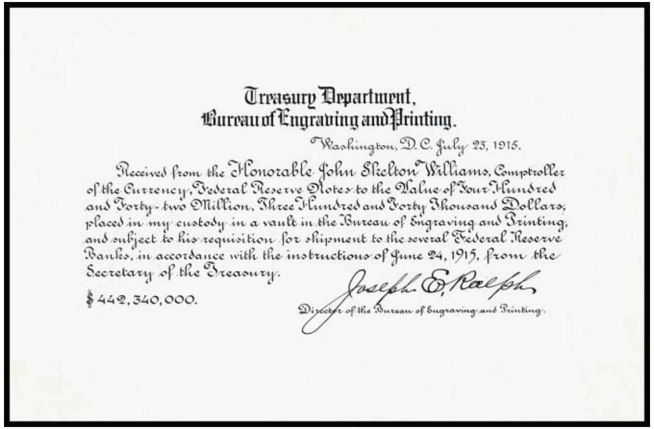
It is increasingly important to manage ***trust*** in *database* and data *analytics*

Motivation

Data is stored as facts in relational databases.

The question is are they trustworthy?

Name	Date	Amount
John Williams	1915-07-23	442,340
Burger Ville	2009-09-15	11.28
Burger Ville	2009-09-15	6.33



Treasury Department,
Bureau of Engraving and Printing.

Washington, D.C. July 23, 1915.

Received from the Honorable John Skelton Williams, Comptroller of the Currency, Federal Reserve Notes to the Value of Four Hundred and Forty-two Million, Three Hundred and Forty Thousand Dollars, placed in my custody in a vault in the Bureau of Engraving and Printing, and subject to his requisition for shipment to the several Federal Reserve Banks, in accordance with the instructions of June 24, 1915, from the Secretary of the Treasury.

$442,340,000.

Joseph E. Ralph
Director of the Bureau of Engraving and Printing.

(John Williams, 1915-07-23, $442,340)

(BurgerVille, 2009-09-15, $11.28)
(BurgerVille, 2009-09-15, $6.33)

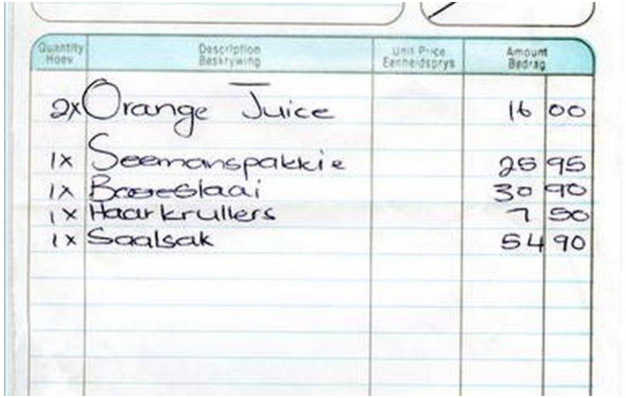
Gaining Trust: an example

Existing methods in establishing (un)trusted data in practice.

Reimbursement for travel expenses.

Gaining Trust: an example

Hand written receipt with *pencil* on *paper*.



Scheme A

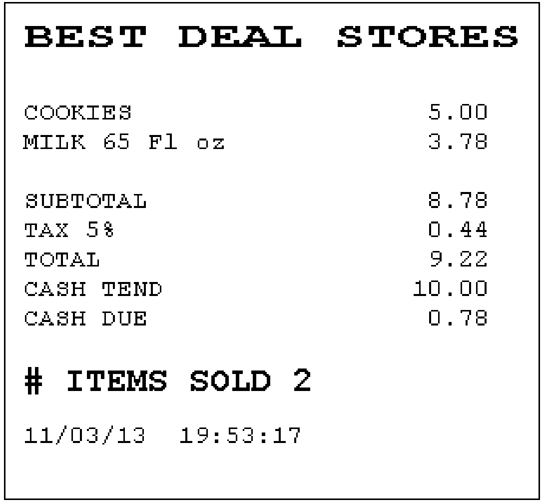
Database technology

1. Easy to create
2. Easy to alter (Mutable)
3. Easy to duplicate with forgery
4. Not verifiable

SQLite database file

Gaining Trust: an example

Printed receipt on business ledger


BEST DEAL STORES

COOKIES	5.00
MILK 65 Fl oz	3.78
SUBTOTAL	8.78
TAX 5%	0.44
TOTAL	9.22
CASH TEND	10.00
CASH DUE	0.78

 # ITEMS SOLD 2

11/03/13 19:53:17

Scheme B

Database technology

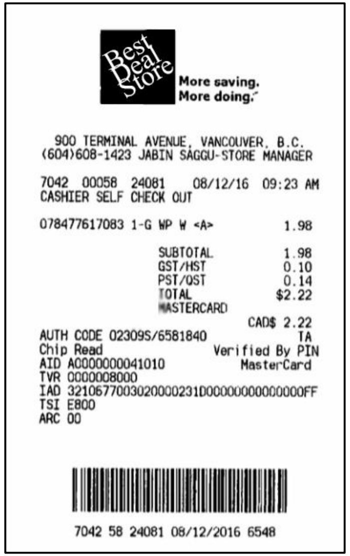
1. Less easy to create
2. Hard to alter (almost immutable)
3. Easy to duplicate with forgery
4. Not verifiable

SQL server with login credentials

Gaining Trust: an example

Printed receipt on business ledger with

1. Signature and Serial number.


Best Deal Store
More saving.
More doing.

900 TERMINAL AVENUE, VANCOUVER, B.C.
(604)608-1423 JABIN SAGGU-STORE MANAGER

7042 00058 24081 08/12/16 09:23 AM
CASHIER SELF CHECK OUT

078477617083 1-G WP W <A> 1.98

SUBTOTAL 1.98
GST/HST 0.10
PST/QST 0.14
TOTAL $2.22
MASTERCARD
CAD$ 2.22
AUTH CODE 02309S/6581840 TA
Chip Read Verified By PIN
AID A0000000041010 MasterCard
TVR 0000008000
IAD 32106770030200002310000000000000FF
TSI E800
ARC 00

7042 58 24081 08/12/2016 6548

Database technology

1. Harder to create
2. Immutable
3. Hard to duplicate with forgery
4. Verifiable (but difficult) using replicated storage

Distributed SQL server (in the cloud)

Scheme C

Gaining Trust: an example

Only business executives can submit expense claims for reimbursement.

NO RECEIPT

Scheme D

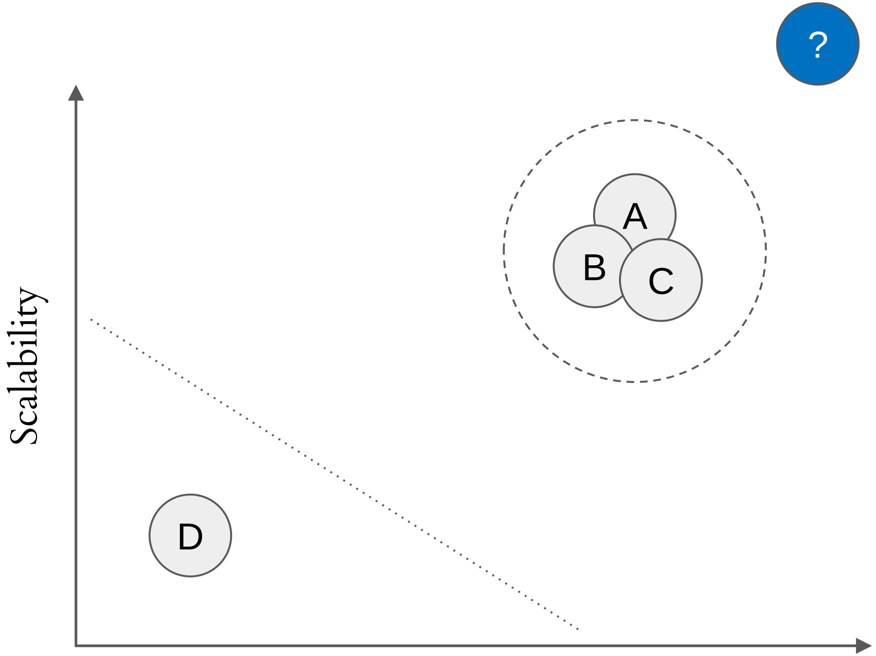
Database technology –

(implicit trust in data)

1. Creation doesn't matter
2. Mutability doesn't matter
3. Difficulty in forgery – N/A
4. Verifiability – N/A

SQL Server with authentication and access control

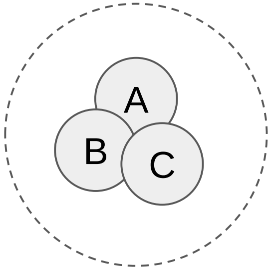
Gaining Trust: an example

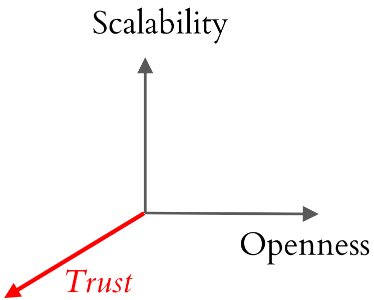


Our research interest is

Scalable and Open Trusted Databases

Gaining Trust: an example

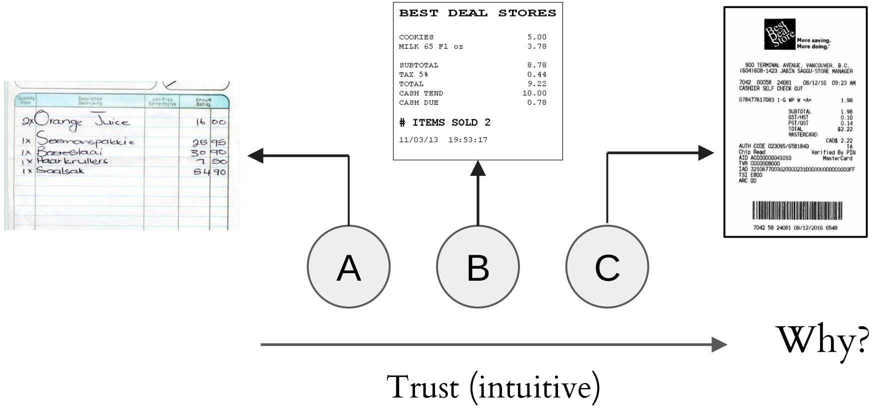




Our research interest is

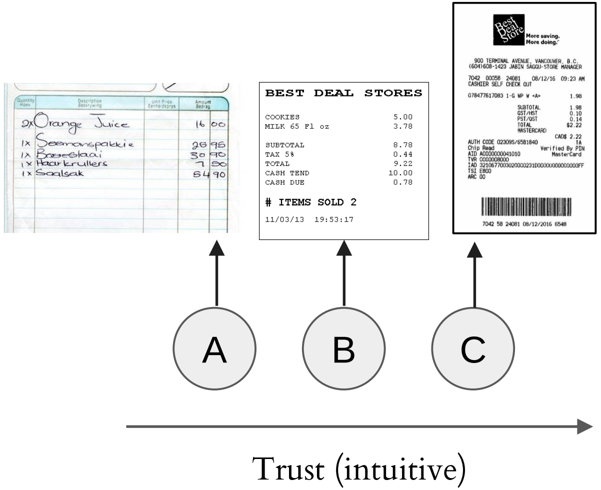
Scalable and Open

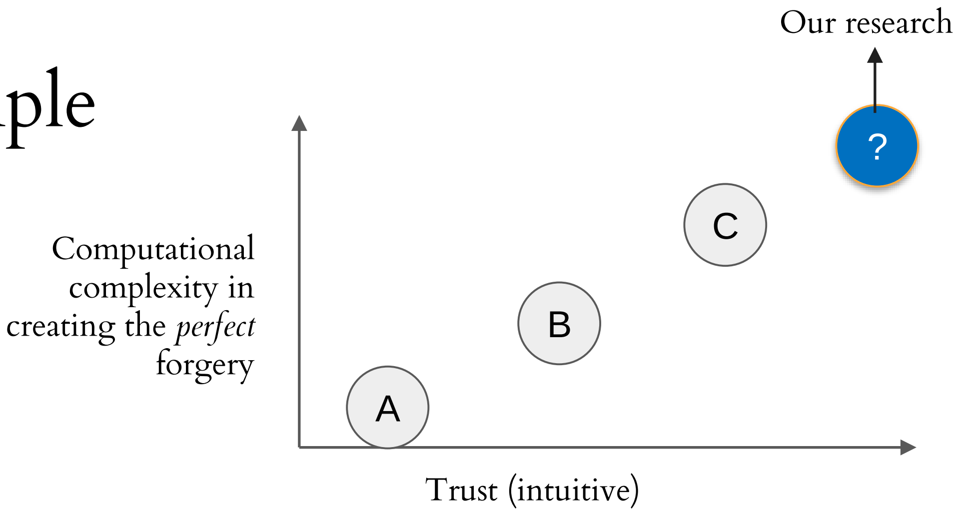
& *Trusted* Databases

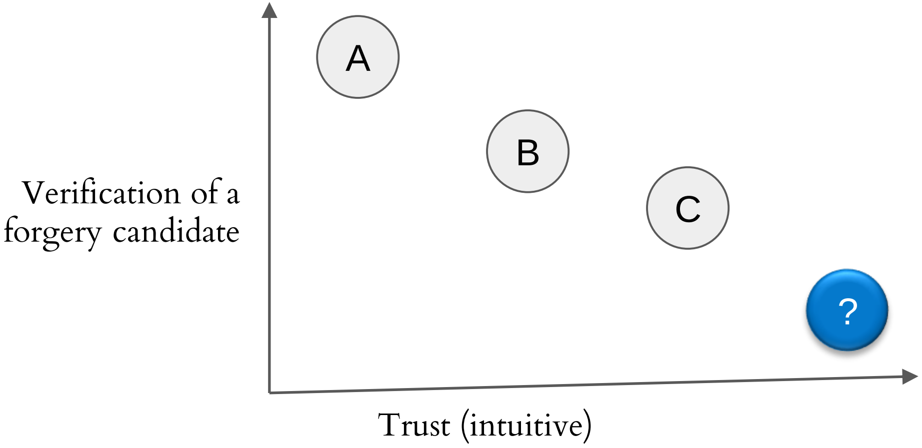


Why?

Gaining Trust: an example







Research

A *trusted* transaction logfile to support:

1. Provide auditable data provenance.
2. Temporal relational snapshots at user defined timestamps.

Part 1
Providing trusted data provenance

Logfile to support data provenance

id	Part Name	Manufacturer	Price
010	Console	Nintendo	399
012	Keyboard	Microsoft	89
⋮	⋮	⋮	⋮

Data provenance

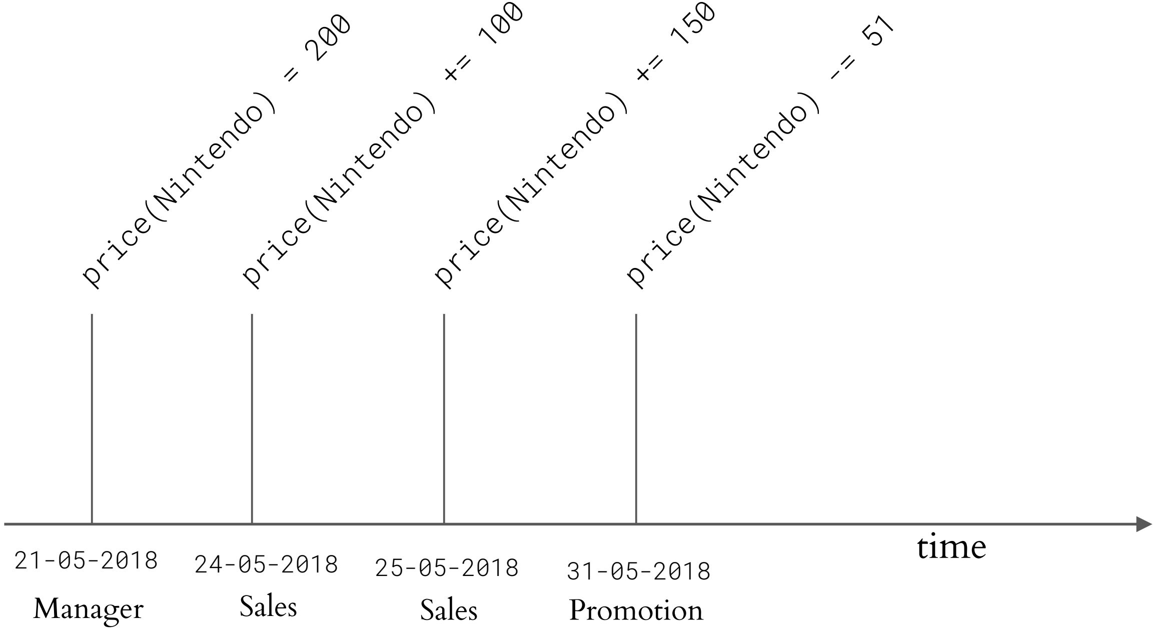
Why is the *price* of *Nintendo* set to $399?

When is the *price* of *Nintendo* set to $399?

Who set the *price* of *Nintendo to* $399?

We need all the historic transactions on the table.
Logfile = **temporal relations**

Logfile to support data provenance



Definition: *temporal relations*

Given a relation r, with attributes $attr(r)$, its temporal relation, written, r^T, *is relation with schema:*

$$attr(r^T) = attr(r) \cup \{\texttt{timestamp}, \texttt{deleted}\}$$

Temporal tables

id	Part Name	Manufacturer	Price
010	Console	Nintendo	399
012	Keyboard	Microsoft	89
. . .	. . .	. . .	. . .

Relational Table

id	Part Name	Manufacturer	Price	timestamp	deleted
010	Console	Nintendo	200	21-05-2018	False
011	Monitor	Asus	499	23-05-2018	False
012	Keyboard	Microsoft	119	23-05-2018	False
010	Console	Nintendo	300	24-05-2018	False
011	-	-	-	25-05-2018	True
012	Keyboard	Microsoft	89	25-05-2018	False
010	Console	Nintendo	450	27-05-2018	False
010	Console	Nintendo	399	31-05-2018	False
. .	. .	. .	. .	. .	. .

Temporal Table

Untrusted provenance

Rec_id	Part Name	Manufacturer	Price	timestamp	deleted
010	Console	Nintendo	200	21-05-2018	False
011	Monitor	Asus	499	23-05-2018	False
012	Keyboard	Microsoft	119	23-05-2018	False
010	Console	Nintendo	300	24-05-2018	False
011	-	-	-	25-05-2018	True
012	Keyboard	Microsoft	89	25-05-2018	False
010	Console	Nintendo	450	27-05-2018	False
010	Console	Nintendo	399	31-05-2018	False
. .	. .	. .	. .	. .	. .

Who committed the transactions?

How can we trust the transactions?

`price(Nintendo) = $399`

Untrusted provenance

Rec_id	Part Name	Manufacturer	Price	timestamp	deleted
010	Console	Nintendo	200	21-05-2018	False
011	Monitor	Asus	499	23-05-2018	False
012	Keyboard	Microsoft	119	23-05-2018	False
010	Console	Nintendo	300	24-05-2018	False
011	-	-	-	25-05-2018	True
012	Keyboard	Microsoft	89	25-05-2018	False
010	Console	Nintendo	450	27-05-2018	False
010	Console	Nintendo	~~470~~ 399	31-05-2018	False
. .	. .	. .	. .	. .	. .

The temporal relation supports data manipulation expressions.

Adversarial attack by super-user.

We need the data to be immutable and verifiable.

`price(Nintendo) = $399`

Temporal database with security information

Given a temporal relation r^T, the trusted version of the relation written, r^{T*}*, is relation with schema:*

$$attr(r^{T*}) = attr(r^T) \cup \{\texttt{username}, \texttt{signature}\}$$

Digital Signature

Mathematical scheme for presenting the authenticity of a digital information

Adds verification to the records

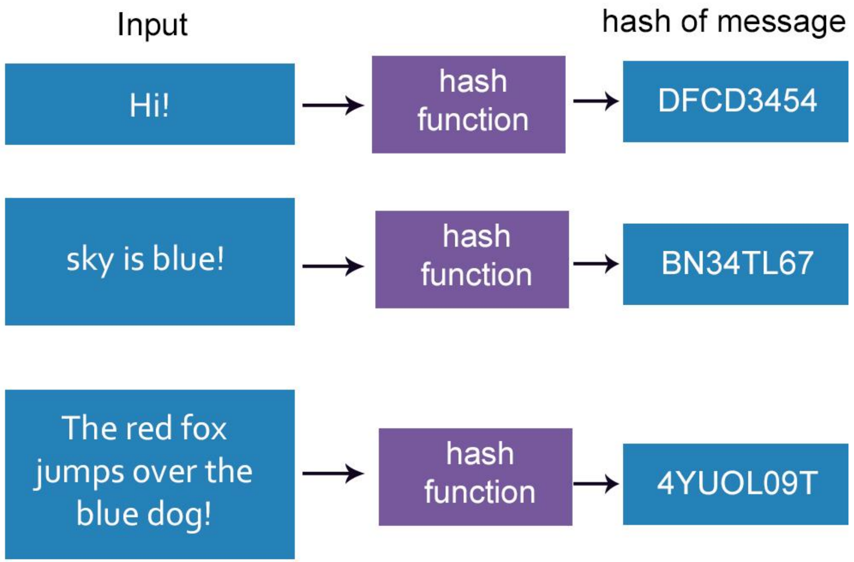
It is unique for user and data

Hash functions

Assume $\boldsymbol{m}$ to be a message of an arbitrary size

$$\boldsymbol{hash(m)} \rightarrow \boldsymbol{sketch}$$

The function maps $\boldsymbol{m}$ from a larger domain to a fixed-size string in a smaller domain



Asymmetric encryption

To encode information such that only authorized parties could read.

Given $\langle K_{private}, K_{public} \rangle \in R^{+}$
$\boldsymbol{K_{private}}$ is known to the owner,
$\boldsymbol{K_{public}}$ is shared with public

cipher = encrypt(m, K_{public})

m = decrypt(cipher, $K_{private}$)

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

Function `encrypt`(*m,publicKey*):
- publicKeyObj = RSA.importKey(publicKey)
- *randomParam* $\leftarrow$ *random.choice*()
- **return** publicKey.encrypt(*m,randomparam*)

Function `decrypt`(*enc_message,privateKey*):
- privateKeyObj = RSA.importKey(privateKey)
- **return** privateKey.decrypt(*enc_message*)

Digital signature

$sig = encrypt(hash(m), K_{private})$

$isValid = verify(m, sig, K_{public})$

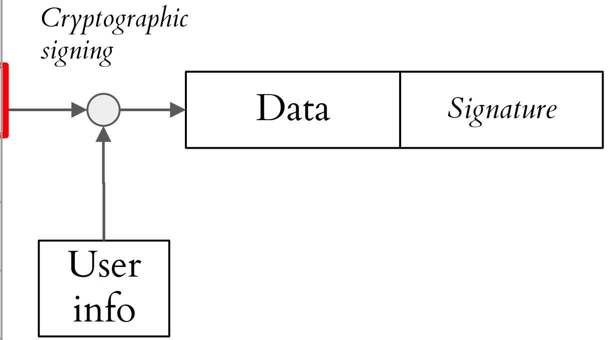
Function `digitalSignature`(m,$privateKey$):
- $hashVal$ = hash(m)
- $randomVal$ = random.choice()
- $privateKeyObj$ = RSA.importKey(privateKey)
- **return** privateKey.encrypt($hashVal$,$randomVal$)

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

Signing transactions

Each user is given a unique ID, and RSA signing key-pairs.

Rec_id	Part Name	Manufacturer	Price	timestamp	deleted
010	Console	Nintendo	200	21-05-2018	False
011	Monitor	Asus	499	23-05-2018	False
012	Keyboard	Microsoft	119	23-05-2018	False
010	Console	Nintendo	300	24-05-2018	False
011	-	-	-	25-05-2018	True
012	Keyboard	Microsoft	89	25-05-2018	False
010	Console	Nintendo	450	27-05-2018	False
010	Console	Nintendo	399	31-05-2018	False
. .	. .	. .	. .	. .	. .



Verifiable without revealing users' secrets

Temporal relations with digital signature

Rec_id	Part Name	Manufacturer	Price	timestamp	deleted	user	sign
010	Console	Nintendo	200	21-05-2018	False	clerk1	6AE2BTN
011	Monitor	Asus	499	23-05-2018	False	manager	NT89RE7
012	Keyboard	Microsoft	119	23-05-2018	False	manager	O4P5KM9
010	Console	Nintendo	300	24-05-2018	False	clerk1	YUI9T35
011	-	-	-	25-05-2018	True	manager	09RTF15
010	Console	Nintendo	450	27-05-2018	False	manager	J3IOGLS
012	Keyboard	Microsoft	89	31-05-2018	False	clerk2	78PP00L
. .	. .	. .	. .	. .	. .		

We know who committed the transactions

Authenticity of the records are verifiable

price(Nintendo) = $450

Temporal relations with digital signature

Rec_id	Part Name	Manufacturer	Price	timestamp	deleted	user	sign
010	Console	Nintendo	200	21-05-2018	False	clerk1	6AE2BTN
011	Monitor	Asus	499	23-05-2018	False	manager	NT89RE7
012	Keyboard	Microsoft	119	23-05-2018	False	manager	O4P5KM9
010	Console	Nintendo	300	24-05-2018	False	clerk1	YUI9T35
011	-	-	-	25-05-2018	True	manager	09RTF15
010	Console	Nintendo	450	27-05-2018	False	manager	J3IOGLS
010	Console	Nintendo	399	29-05-2018	False	clerk2	SDN0PLT
012	Keyboard	Microsoft	89	31-05-2018	False	clerk2	78PP00L
. .	. .	. .	. .	. .	. .		

Adversarial attack is more difficult now, but still possible with committing malicious transactions.

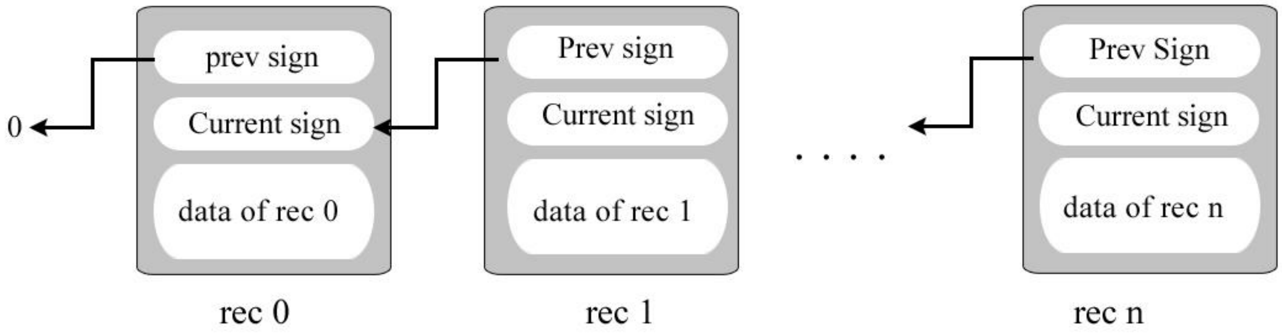
Clerk2 is a superuser

`price(Nintendo) = $399`

Blockchain

Set of blocks chained together by *digital signatures*.

Each block contains its previous block's digital signature



Temporal database with security information

Given a temporal relation with security information r^{T*}, the relation with chained records, written as, r^{T+} has the schema:

$$attr(r^{T+}) = attr(r^{T*}) \cup \{\texttt{previous\_signature}\}$$

Trusted temporal tables with *blockchain*

Rec_id	Part Name	Manufacturer	Price	timestamp	deleted	user	sign	pre_sgn
010	Console	Nintendo	200	21-05-2018	False	clerk1	6AE2BTN	0
011	Monitor	Asus	499	23-05-2018	False	manager	NT89RE7	6AE2BTN
012	Keyboard	Microsoft	119	23-05-2018	False	manager	O4P5KM9	NT89RE7
010	Console	Nintendo	300	24-05-2018	False	clerk1	YUI9T35	O4P5KM9
011	-	-	-	25-05-2018	True	manager	09RTF15	YUI9T35
012	Keyboard	Microsoft	89	25-05-2018	False	manager	J3IOGLS	09RTF15
010	Console	Nintendo	450	27-05-2018	False	clerk2	78PP00L	J3IOGLS
010	Console	Nintendo	399	31-05-2018	False	manager	ZNTP67W	78PP00L
. .	. .	. .	. .	. .	. .			

Trusted temporal tables with *blockchain*

Rec_id	Part Name	Manufacturer	Price	timestamp	deleted	user	sign	pre_sgn
010	Console	Nintendo	200	21-05-2018	False	clerk1	6AE2BTN	0
011	Monitor	Asus	499	23-05-2018	False	manager	NT89RE7	6AE2BTN
012	Keyboard	Microsoft	119	23-05-2018	False	manager	O4P5KM9	NT89RE7
010	Console	Nintendo	~~300~~ 100	24-05-2018	False	clerk1	YUI9T35	O4P5KM9
011	-	-	-	25-05-2018	True	manager	09RTF15	YUI9T35
012	Keyboard	Microsoft	89	25-05-2018	False	manager	J3IOGLS	09RTF15
010	Console	Nintendo	450	27-05-2018	False	clerk2	78PP00L	J3IOGLS
010	Console	Nintendo	399	31-05-2018	False	manager	ZNTP67W	78PP00L
. .	. .	. .	. .	. .	. .			

*Adversarial attack by **clerk2**.*

Trusted temporal tables with *blockchain*

Rec_id	Part Name	Manufacturer	Price	timestamp	deleted	user	sign	pre_sgn
010	Console	Nintendo	200	21-05-2018	False	clerk1	6AE2BTN	0
011	Monitor	Asus	499	23-05-2018	False	manager	NT89RE7	6AE2BTN
012	Keyboard	Microsoft	119	23-05-2018	False	manager	O4P5KM9	NT89RE7
010	Console	Nintendo	100	24-05-2018	False	Clerk 2	OPNJNP3	O4P5KM9
011	-	-	-	25-05-2018	True	manager	09RTF15	YUI9T35
012	Keyboard	Microsoft	89	25-05-2018	False	manager	J3IOGLS	09RTF15
010	Console	Nintendo	450	27-05-2018	False	clerk2	78PP00L	J3IOGLS
010	Console	Nintendo	399	31-05-2018	False	manager	ZNTP67W	78PP00L
. .	. .	. .	. .	. .	. .			

Adversarial attack by ***clerk2.***

Broken chain

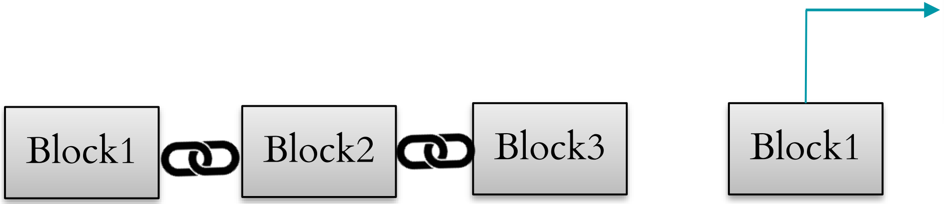
Trusted temporal tables with *blockchain*

Rec_id	Part Name	Manufacturer	Price	timestamp	deleted	user	sign	pre_sgn
010	Console	Nintendo	200	21-05-2018	False	clerk1	6AE2BTN	0
011	Monitor	Asus	499	23-05-2018	False	manager	NT89RE7	6AE2BTN
012	Keyboard	Microsoft	119	23-05-2018	False	manager	O4P5KM9	NT89RE7
010	Console	Nintendo	100	24-05-2018	False	Clerk 2	OPNJNP3	O4P5KM9
011	-	-	-	25-05-2018	True	Clerk 2	NYTO903	OPNJNP3
012	Keyboard	Microsoft	89	25-05-2018	False	Clerk 2	YUI89PY	NYTO903
010	Console	Nintendo	450	27-05-2018	False	Clerk 2	09SSTER	YUI89PY
010	Console	Nintendo	399	31-05-2018	False	Clerk 2	IOP7TU8	09SSTER
. .	. .	. .	. .	. .	. .			

Adversarial attack by ***clerk2.***

Adding records to the Blockchain

- Collect the timestamp
- Sign block 1
- Add user info
- Store the signature of previous block



Adding blocks could be done with O(1)

Verifying Blockchain

```
Function blockchainVerification(last_id):
    for i ← 1 to last_id do
        publicKey ← SELECT publicKey FROM USERS WHERE user = username;
        validity = verifySign(rec_i,signature,publicKey)
        if validity != valid then
            return Broken chain
            Break
        if rec_i[prev_sgn] != rec_{i-1}[signature] then
            return Broken chain
            Break
    end
    return Valid Chain
```

Part 2
Query Evaluation Over Trusted Temporal Relation

Temporal databases: snapshot creation

Rec_id	Part Name	Manufacturer	Price	timestamp	deleted	user	sign	pre_sgn
010	Console	Nintendo	200	21-05-2018	False	clerk1	6AE2BTN	0
011	Monitor	Asus	499	23-05-2018	False	manager	NT89RE7	6AE2BTN
012	Keyboard	Microsoft	119	23-05-2018	False	manager	O4P5KM9	NT89RE7
010	Console	Nintendo	300	24-05-2018	False	clerk1	YUI9T35	O4P5KM9
011	-	-	-	25-05-2018	True	manager	09RTF15	YUI9T35
012	Keyboard	Microsoft	89	25-05-2018	False	manager	J3IOGLS	09RTF15
010	Console	Nintendo	450	27-05-2018	False	clerk2	78PP00L	J3IOGLS
010	Console	Nintendo	399	31-05-2018	False	manager	ZNTP67W	78PP00L
. .	. .	. .	. .	. .	. .			

What did the relation/record look like in 25-05-2018?

Are the records/ snapshots until then trustworthy?

Temporal databases: snapshot creation

We can generate snapshots using relational database windowing functions

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

Temporal databases: snapshot creation

Rec_id	Part Name	Manufacturer	Price	timestamp	deleted	user	sign	pre_sgn
010	Console	Nintendo	200	21-05-2018	False	clerk1	6AE2BTN	0
011	Monitor	Asus	499	23-05-2018	False	manager	NT89RE7	6AE2BTN
012	Keyboard	Microsoft	119	23-05-2018	False	manager	O4P5KM9	NT89RE7
010	Console	Nintendo	300	24-05-2018	False	clerk1	YUI9T35	O4P5KM9
011	-	-	-	25-05-2018	True	manager	09RTF15	YUI9T35
012	Keyboard	Microsoft	89	25-05-2018	False	manager	J3IOGLS	09RTF15
010	Console	Nintendo	450	27-05-2018	False	clerk2	78PP00L	J3IOGLS
010	Console	Nintendo	399	31-05-2018	False	manager	ZNTP67W	78PP00L
. .	. .	. .	. .	. .	. .			

Records to be processed.

Temporal databases: snapshot creation

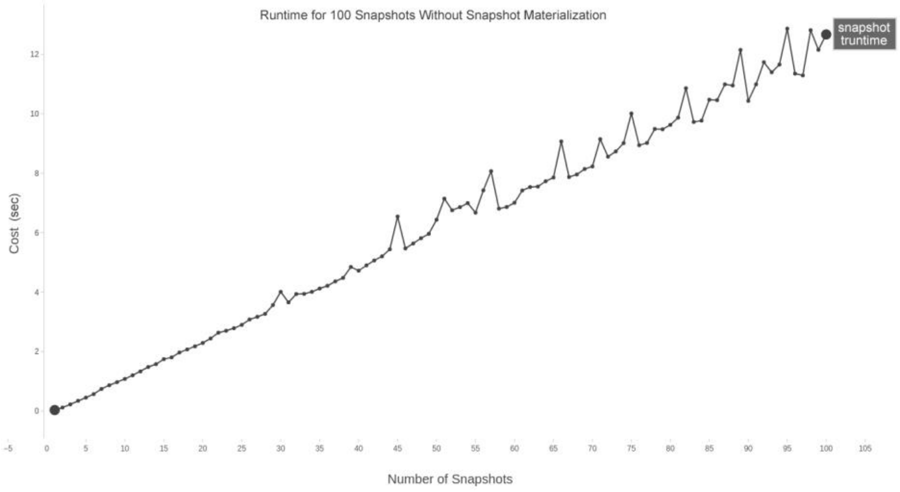
The snapshot creation and blockchain verification take linear time.

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

Recomputing snapshots and blockchain verification is expensive.

Temporal databases: snapshot creation

Runtime of creating 1-100 snapshots



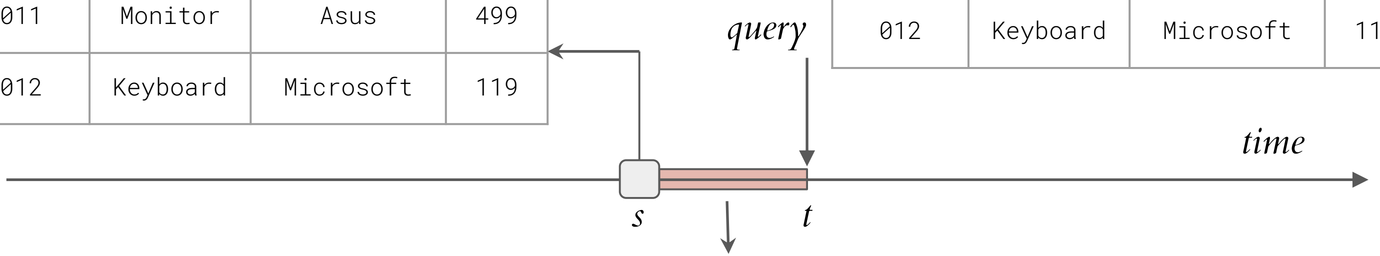
Lowering the cost of snapshot generation

We propose to *precompute* snapshots for *materialization* to lower the cost of snapshot generation.

Snapshot

id	Part Name	Manufacturer	Price
010	Console	Nintendo	200
011	Monitor	Asus	499
012	Keyboard	Microsoft	119

id	Part Name	Manufacturer	Price
010	Console	Nintendo	399
012	Keyboard	Microsoft	119



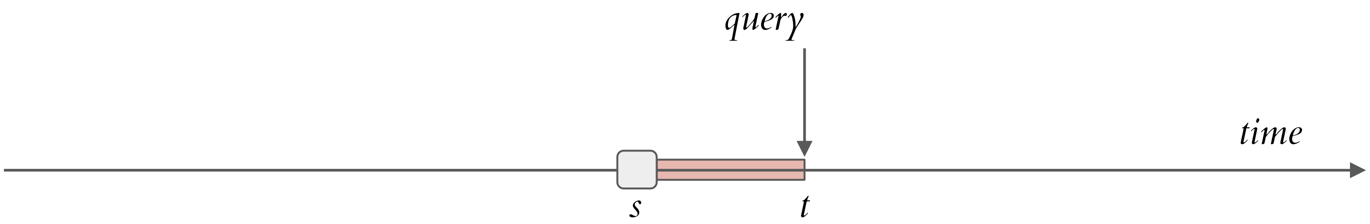
(10, console, Nintendo, 300, False, 24-05-2018, clerk1, YUI9T35, 04P5KM9)

(11, , , , True, 25-05-2018, manager, 09RTF15, YUI9T35)

(10, console, Nintendo, 399, False, 25-05-2018, clerk2, 78PP00L, 09RTF15)

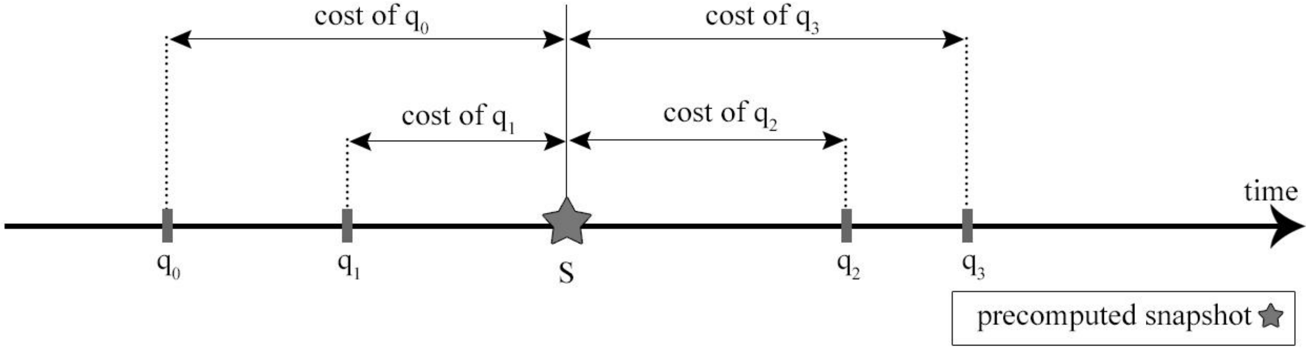
Lowering the cost of snapshot generation

We propose to *precompute* snapshots for *materialization* to lower the cost of snapshot generation.



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

Overall cost of query using a materialized snapshot

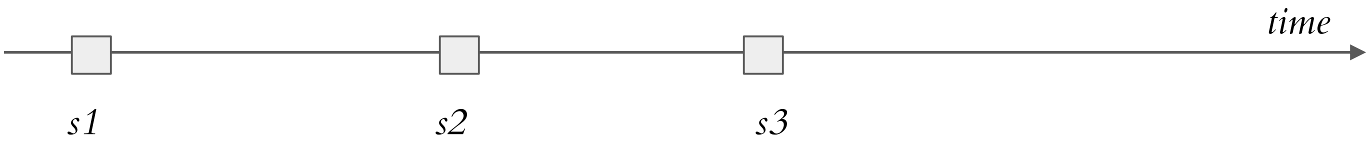


Cost of query using single materialized snapshot is

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

Lowering the cost of snapshot generation

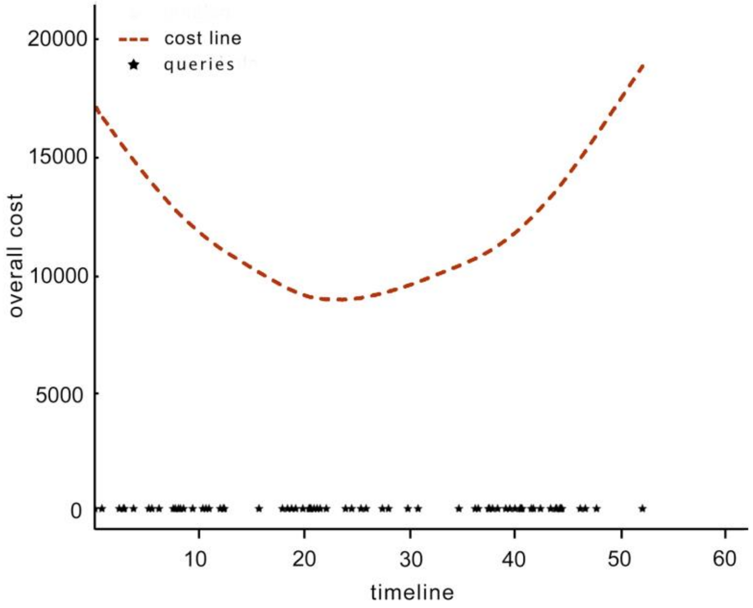
*Our goal is to precompute **multiple optimal** snapshots for materialization*



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

Single snapshot materialization

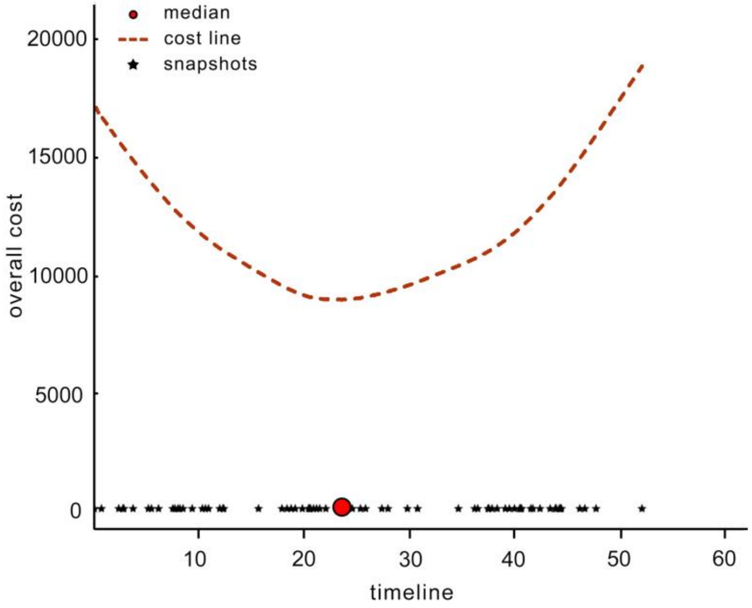
Where is the optimal timestamp for snapshot placement?



Single snapshot materialization

Theorem. The median of queries minimizes the sum of absolute deviation

If the snapshot was in the median of the performed queries, it was in the optimal timestamp.



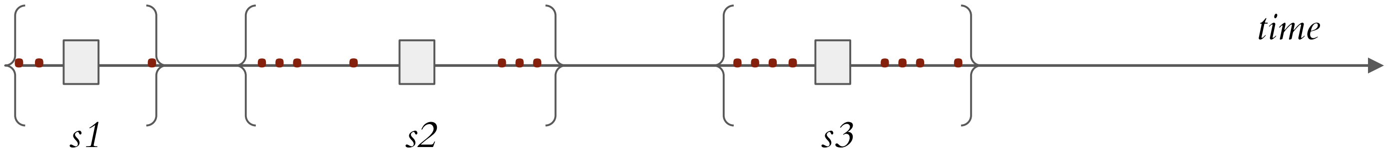
Lowering the cost of snapshot generation

We suggest to store the timestamp of previous queries

It gives us insight into the pattern of queries and hotspots of the timeline

Lowering the cost of snapshot generation

How can we create optimal segmentations in which if a snapshot is placed for materialization, the overall cost of query answering is minimum?



Multiple optimal snapshot placement: possible solutions

The segmentations could be computed with:

1. Recursive algorithm
2. Dynamic programming
3. Heuristic method

Which one is better?

Optimal query segmentation: Recursive Algorithm

We try all the last segment of the queries and we choose the one with the lowest cost.

Benefits:
The optimal solution is guaranteed

Drawbacks:
Inefficiency in computation for large number of queries

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**
      $j^* = \underset{j \in [1,i]}{\operatorname{argmin}}(\operatorname{cost}(\mathbf{OPT}[j,k-1]) + \operatorname{cost}(Q[j+1,n]))$
      $\mathbf{OPT}[i,k] = \mathbf{OPT}[j^*,k-1] \cup \{\operatorname{median}(Q[j+1],n)\}$
    end
  end

The recursive algorithm requires $\mathcal{O}(2^m)$

Optimal query segmentation: Dynamic Programming

The result of expensive function calls are stored in a table, and retrieved when the same input is given to the function.

Benefits:
The optimal solution is guaranteed and has lower cost than recursion.

Drawbacks:
Inefficiency in computation for larger number of queries

```
Function computeOPT(Q, m):
    n = |Q|
    minVal = ∞
    for i ← 1 to m do
        for j ← 1 to n + 1 do
            for k ← 1 to j do
                minVal = min(minVal, Table[i, k] + cost(Q[j − k :]))
            end
            Table[i, j] = minVal
        end
    end
```

The dynamic programming requires $\mathcal{O}(mn^2)$

Optimal query segmentation: Heuristic method

Random timestamps are chosen and partitions are created. Then the random timestamps are moved on the timeline until the within cluster sum of squares is minimized.

Benefits:
Low time complexity

Drawbacks:
The optimal solution not guaranteed.

The K-means clustering algorithm is given as Algorithm 9.

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** $\{\mu_1, ..., \mu_m\}$

The K-Means clustering requires $\mathcal{O}(m * k * n)$

Performance of optimal snapshot computation: experiment

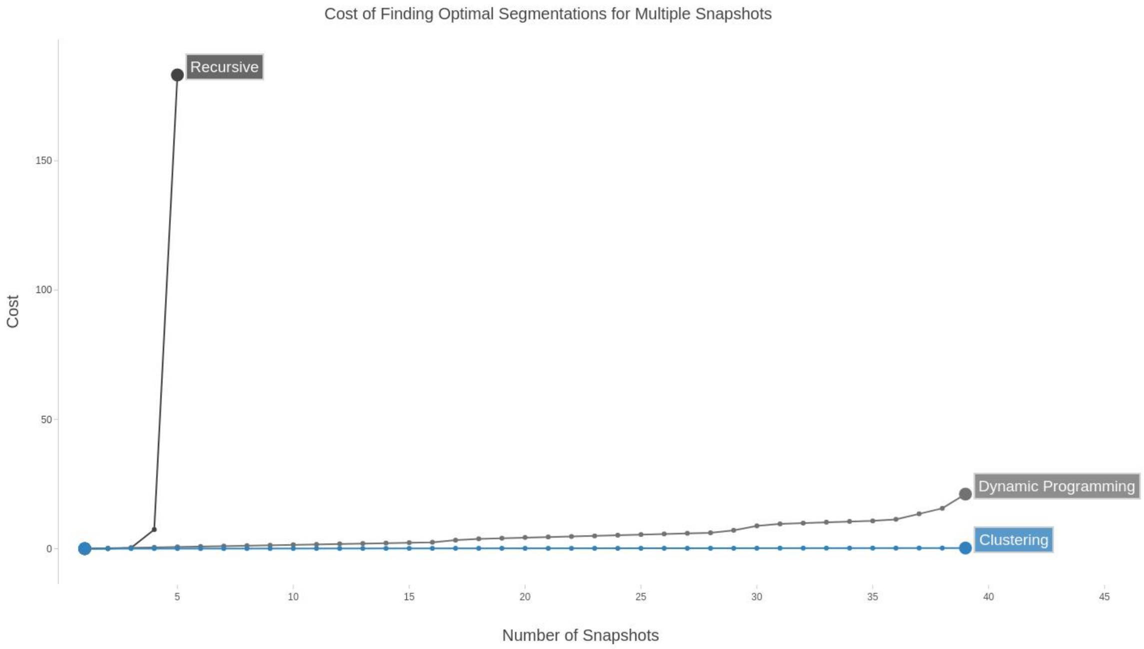
Experiment 1:

Fixed number of queries: 140
variable number of snapshots: 1 - 40

Objective:

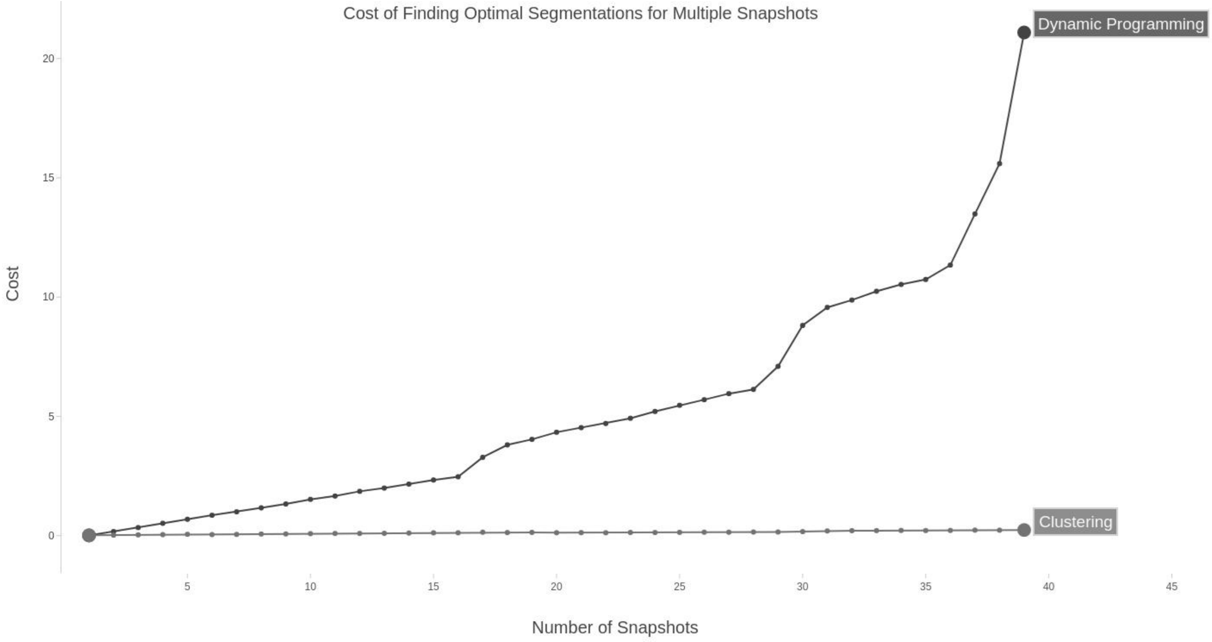
Evaluate performance of algorithms with respect to our project

performance of optimal snapshot computation :comparison



Snapshots	Recursive	Dynamic	Clustering
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

Performance of optimal snapshot computation :comparison

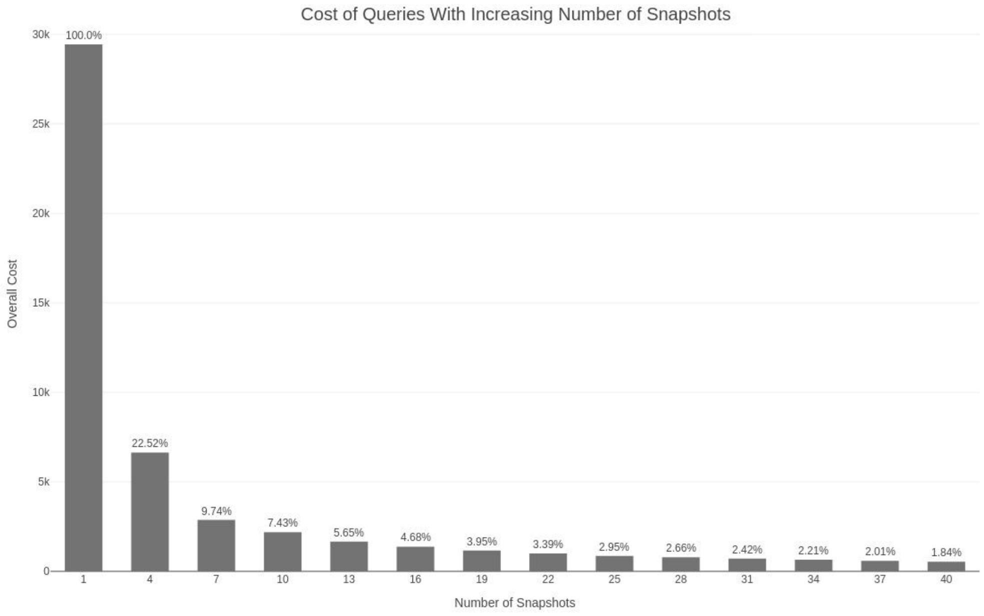


Snapshots	Dynamic	Clustering
1	0.01	0.01
2	0.17	0.02
3	0.34	0.03
4	0.52	0.04
5	0.69	0.06
10	1.52	0.08
15	2.33	0.12
20	4.33	0.12
25	5.46	0.14
30	8.81	0.17
35	10.74	0.21
40	21.09	0.24

Experiment 1. Effectiveness of snapshot materialization

Case: The optimal segmentations computed using dynamic programming

Objective: To see how increasing number of materialized snapshot affects the cost



Performance of optimal snapshot computation: experiment

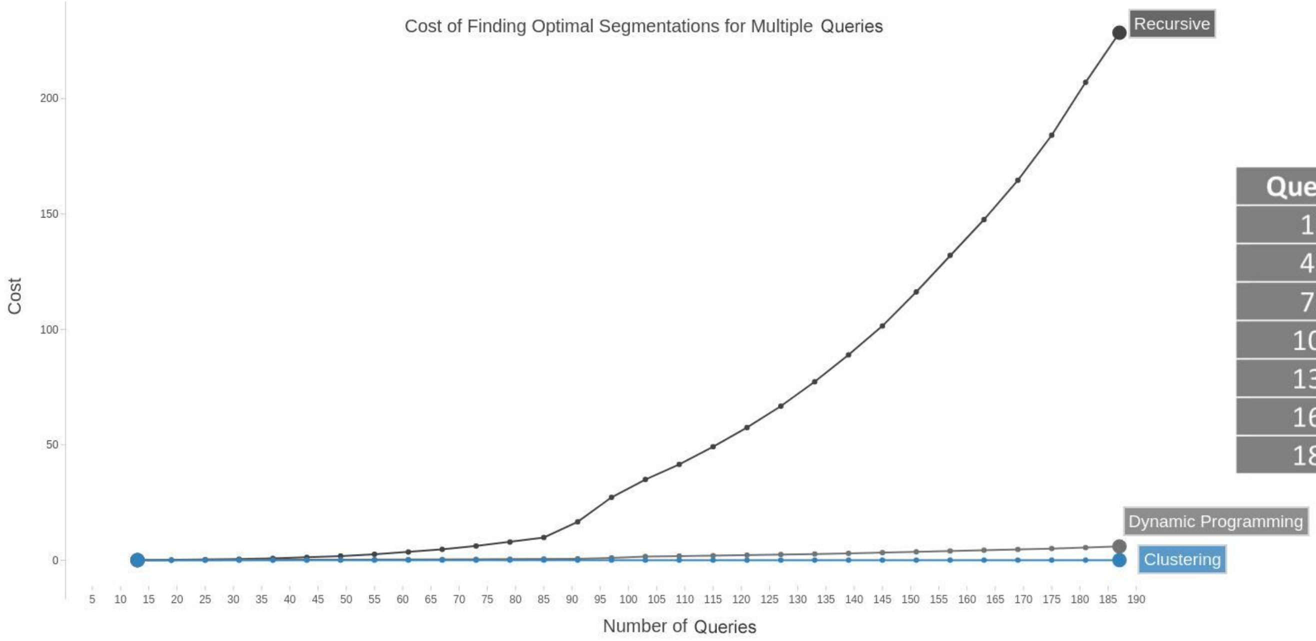
Experiment 2:

Fixed number of snapshots: 4
variable number of queries: 12 - 190

Objective:

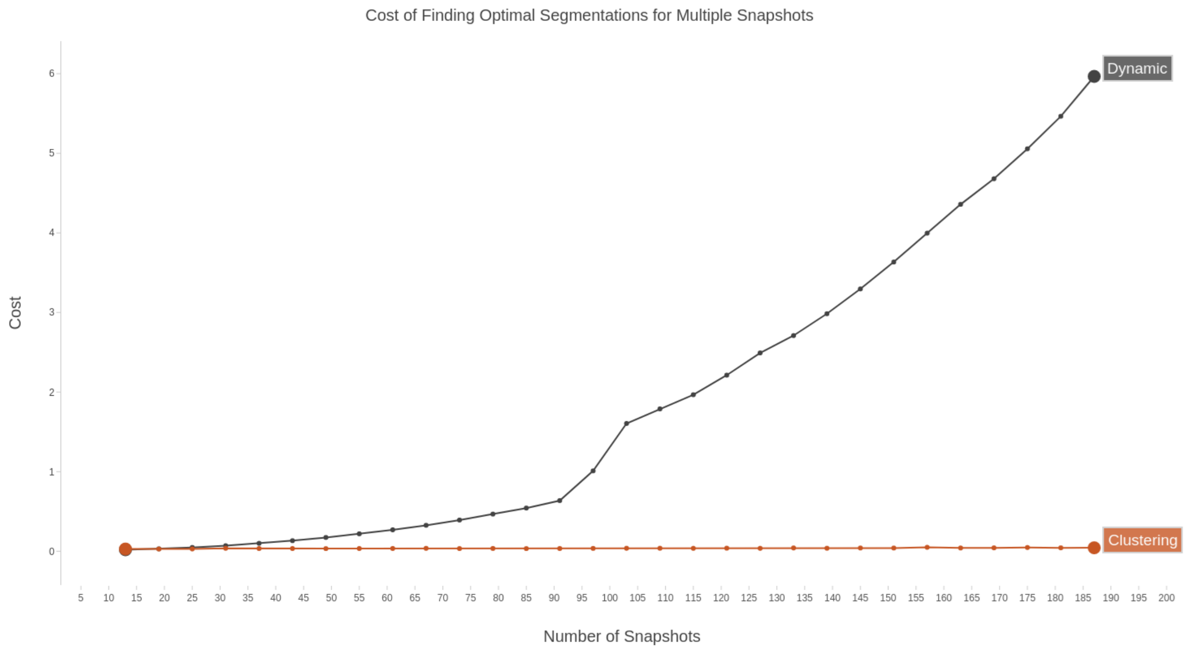
Evaluate performance of algorithms with respect to our project

Performance of optimal snapshot computation: comparison



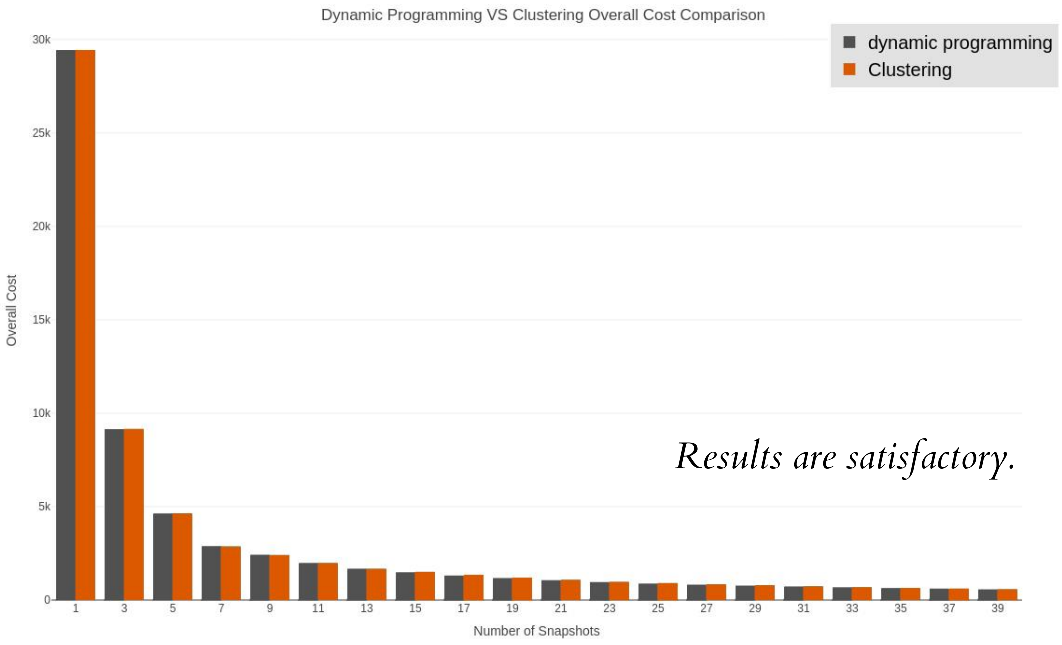
Queries	Recursive	Dynamic	Clustering
13	0.05	0.01	0.02
43	1.23	0.13	0.03
73	6.15	0.39	0.03
103	34.98	1.60	0.03
133	77.31	2.71	0.04
163	147.57	4.35	0.04
187	228.52	5.96	0.04

Performance of optimal snapshot computation: comparison



Queries	Dynamic	Clustering
13	0.01	0.02
43	0.13	0.03
73	0.39	0.03
103	1.60	0.03
133	2.71	0.04
163	4.35	0.04
187	5.96	0.04

Performance of optimal snapshot computation: comparison

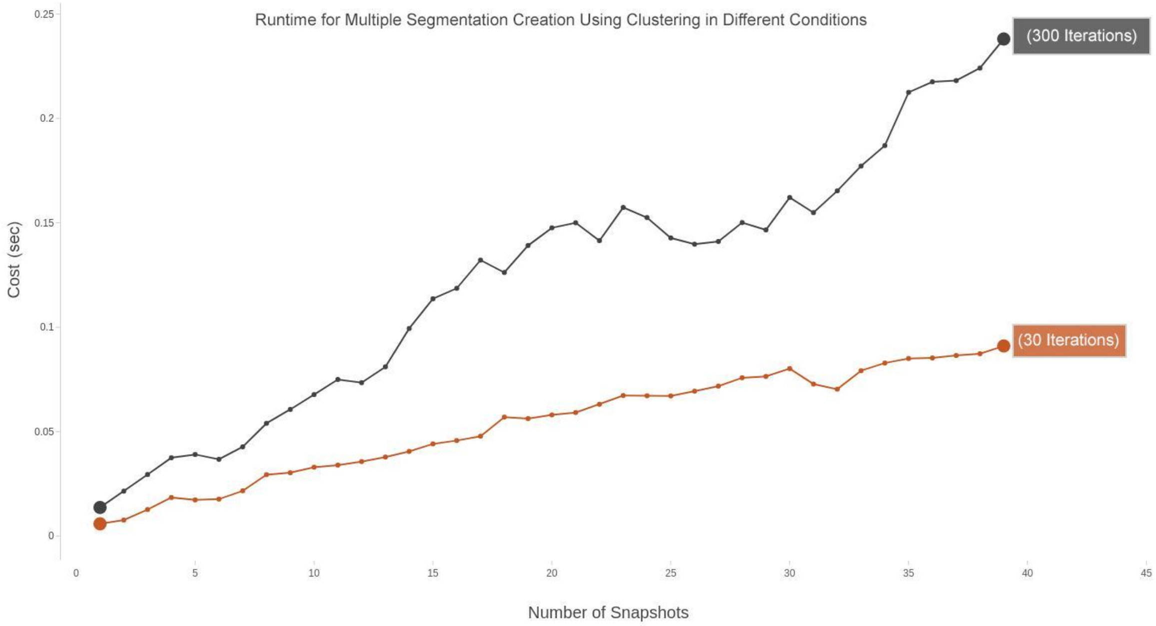


Results are satisfactory.

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	1079.711
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

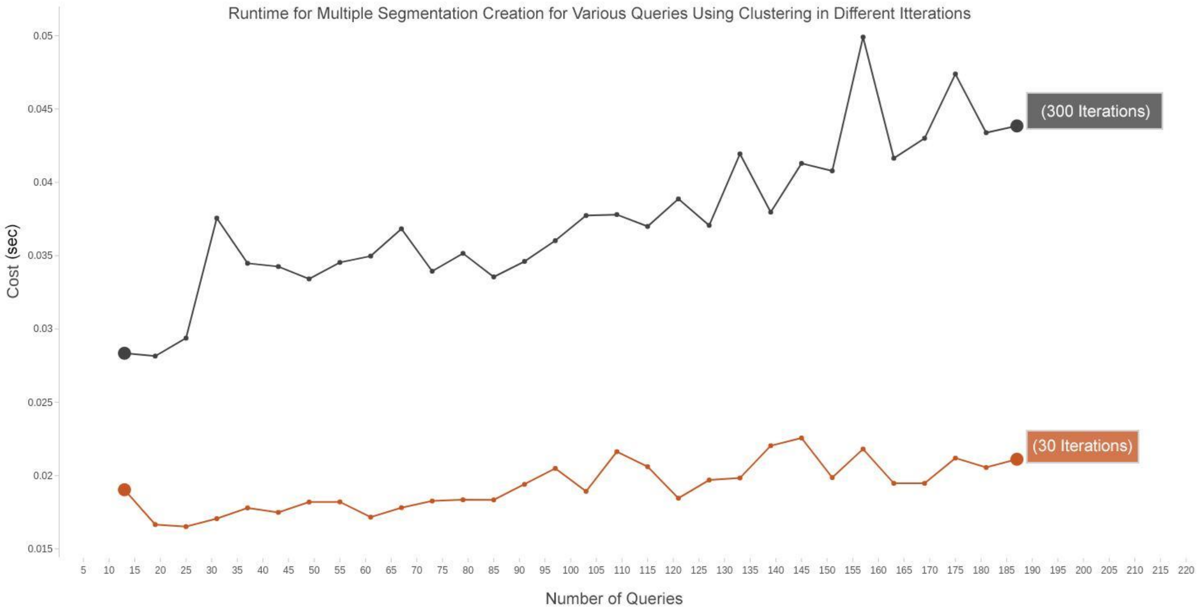
Performance of optimal snapshot computation: comparison

Runtime for variable number of snapshots but fixed number of queries



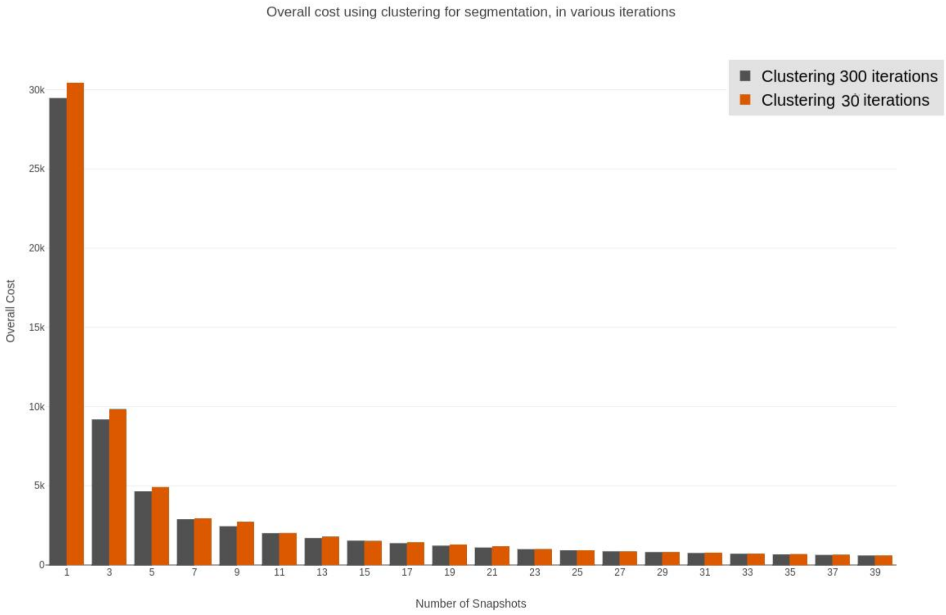
Performance of optimal snapshot computation: comparison

Runtime for fixed number of queries but variable number of snapshots



Performance of optimal snapshot computation: comparison

Comparison between the overall cost of query answering using optimal segmentations made by heuristic method in different iterations



Performance of optimal snapshot computation: experiment

Experiment 3:

40 snapshots with multiple segmentation strategy on a timeline with **1000** queries.

Objective:

To see if finding optimal segmentations worth it.

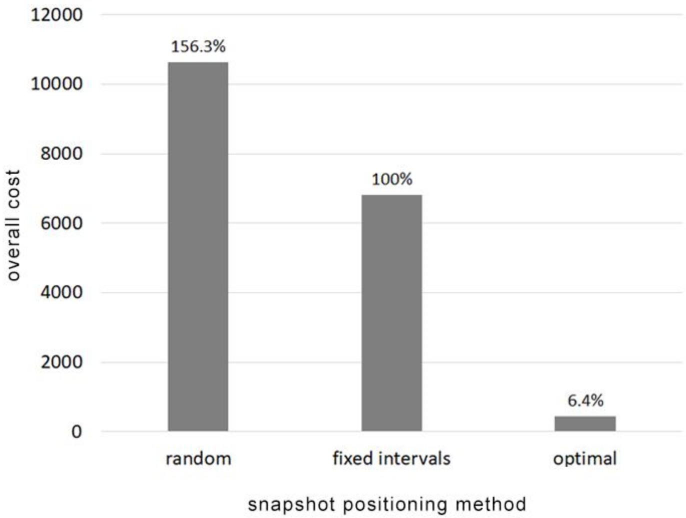
Experiment 1. Effectiveness of snapshot materialization

Case 1: Segmentations created randomly

Case 2: Fixed segmentations were created

Case 3: Optimal segmentation was computed using dynamic programming

Objective: Does optimal segmentation computation make any difference? YES

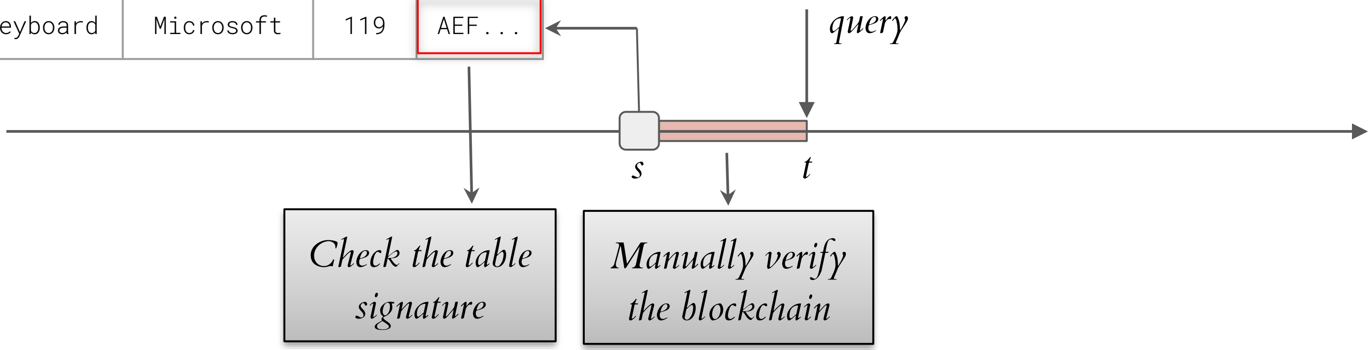


Further remarks

Lowering the cost of blockchain verification

We propose to *digitally sign* the *precompute* snapshots.

id	Part Name	Manufacturer	Price	tbl_sgn
010	Console	Nintendo	200	-
011	Monitor	Asus	499	-
012	Keyboard	Microsoft	119	AEF...

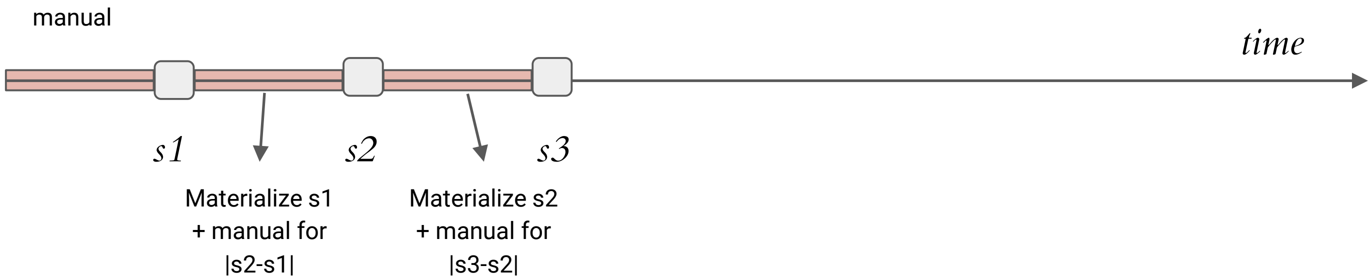


Materialized snapshot generation

How can we reduce the cost of snapshot generation?

For the first snapshot, computation is done manually.

For the subsequent snapshots, previous snapshots are materialized.

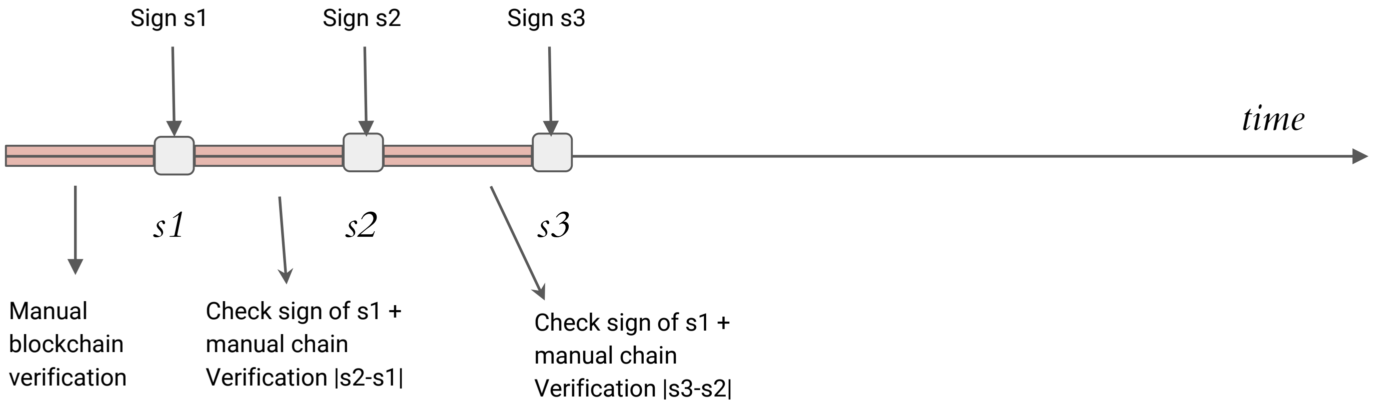


Materialized Blockchain verification

For the first snapshot, Blockchain verification is done manually.

How can we reduce the cost of snapshot signing?

For the subsequent snapshots, previous snapshots are materialized.



Conclusion

- Using the temporal relation that utilizes Blockchain we can provide trusted data provenance.
- Snapshot materialization has promising results in reducing the cost of query answering.
- With storing the timestamp of performed queries we can find the hotspots on the timeline and we can create optimal segmentations that materialize their exclusive snapshot
- Optimal snapshot placement has more impact in reducing cost than random or fixed interval snapshot placement.

Conclusion

- Dynamic programming, recursion and heuristic methods could be used for optimal segmentation computation.
- In this project, the heuristic method is favorable because of efficient cost and satisfactory results.
- To reduce the cost of multiple snapshot placement as well as chain verification, previous snapshots could be materialized.
- Snapshots could be dynamically repositioned as the new queries performed on the system.

Towards Trusted Temporal Databases Using Blockchain

By Amin Beirami
MSc student, UOIT

Supervisors: Dr. Ken Pu and Dr. Ying Zhu

October 2018