Bank Checks Fraud Detection Based on the Analysis of Event Trends in Data-Flow Systems

Uriy Grigorev[1][a], Yury Shashkin[1][b], Andrey Ploutenko[2][c], Aleksey Burdakov[1][d] and Olga Pluzhnikova[1][e]

[1]*Bauman Moscow State Technical University, Moscow, Russia*

[2]*Amur State University, Blagoveschensk, Russia*

Keywords: CEP, Fraud with Not-covered Checks, CET Graph, BTree Index, Hash Index.

Abstract: The paper shows trend analysis of events in data-flow systems on the example of fraud detection with not-covered checks. The analysis is based on Complex Event Processing (CEP) technology. This article proposes Algorithm 2 based on BTree and Hash type indexes for extracting a complete chain of events of any length formed by insufficient funds check deposits. The paper presents a comparison between the proposed Algorithm 2 and the existing Algorithm 1, based on the construction of event trends in the form of graphs. The average processing time of one event using the new Algorithm 2 is 56 times less with the number of events equal to 100,000. At the same time, the new Algorithm 2 processes about 900,000 events, while the existing Algorithm 1 supports only 100,000 events.

1 INTRODUCTION

Large data amounts processing in real time is an important requirement in modern high-load systems. Data-flow processing systems are used to solve this problem. Data-flow processing has found application in retail (Wu et al., 2006), stock exchange (Poppe et al., 2017a; Agrawal et al., 2008), fraud detection (Luckham, 2011; Agrawal et al., 2008; Poppe et al., 2017b), passenger transportation (Luckham, 2011), computer cluster monitoring (Poppe et al., 2017a; Zhang et al., 2014), logistics management (Zhang et al., 2014), traffic congestion detection (Poppe et al., 2017a), video streaming (Piyush et al., 2019) and many other areas.

Nowadays the Complex Event Processing (CEP) technology is used to support streaming applications (Poppe et al., 2017a; Agrawal et al., 2008). CEP systems constantly evaluate events in high-speed streams (data is "passed" through the query). There are two classes of problems solved by CEP. The first class is the aggregation of event trends (calculation of count, avg, sum, min, max by pattern) (Poppe et al., 2017a; Ma et al., 2022), the second class is the analysis of complete event trends (detection of event chains by pattern) (Poppe et al., 2017b; Kolchinsky et al., 2019).

In this article, the problem of event trends analysis is considered with the example of circular check kiting which is used to fraudulently obtain funds (Poppe et al., 2017b). This type of fraud is one of the most difficult.

In a simple case, the scheme involves writing a check for an amount from an account in Bank A that exceeds the account balance; and then writing a check from another account in Bank B which also has insufficient funds. Moreover, the second check serves to cover non-existing funds from the first account. The scammers withdraw funds from the Bank A account before the banks discover the scheme. There were complex variants of this scheme with the participation of numerous scammers who posed as large entrepreneurs, thereby posing their activities as ordinary business transactions. Thus, they persuade

[a] https://orcid.org/0000-0001-6421-3353
[b] https://orcid.org/0000-0001-9576-9119
[c] https://orcid.org/0000-0002-4080-8683
[d] https://orcid.org/0000-0001-6128-9897
[e] https://orcid.org/0000-0002-4276-8734

Grigorev, U., Shashkin, Y., Ploutenko, A., Burdakov, A. and Pluzhnikova, O.
Bank Checks Fraud Detection Based on the Analysis of Event Trends in Data-Flow Systems.
DOI: 10.5220/0011727300003482
In *Proceedings of the 8th International Conference on Internet of Things, Big Data and Security (IoTBDS 2023)*, pages 97-104
ISBN: 978-989-758-643-9; ISSN: 2184-4976
Copyright © 2023 by SCITEPRESS – Science and Technology Publications, Lda. Under CC license (CC BY-NC-ND 4.0)

banks to ignore the limit of available funds. To implement this scheme, fraudsters transfer millions between banks using a complex web of useless checks. For example, in 2014, 12 people were charged with a large-scale fraudulent scheme that cost banks more than $15 million (The Press Enterprise, 2015).

The CEP system is used to prevent this kind of fraud. It constantly monitors events in the flow of financial transactions (Poppe et al., 2017b). The event stream passes through the following query:

```
Q1: PATTERN Check+ c[ ]
WHERE   c.type   =   'notcovered'   AND
c.destination = NEXT(c).source
WITHIN 1 day SLIDE 10 minutes
```

Query Q1 detects a chain (or circle) of any length formed by insufficient funds check deposits during a day that slides every 10 minutes. The query pattern is the Kleene closure (Agrawal et al., 2008; Zhang et al., 2014; Poppe et al., 2017b) for check deposit events, designated as Check+ c[]. Predicates (WHERE) require that the checks in the chain are not covered. The purpose (destination) of transaction verification 'c' should be the same as the source of the next check NEXT(c). It is necessary for identifying the chain. Since an arbitrary number of fraudsters, financial transactions and banks around the world can be involved in this scheme, detecting circular check kiting is a computationally expensive task. To prevent cash withdrawals from an account that participates in at least one check-picking scheme, the query constantly analyzes the streams of high-speed events with thousands of financial transactions per second. It reveals all the trends of receipt compilation in real-time.

The query refers to the 'Skip till any match' type (S3) (Agrawal et al., 2008; Zhang et al., 2014; Volkova et al., 2020; Poppe et al., 2019). This is the most complex class of queries. In the existing methods of solving such problems, Kleene pattern is used to detect the required chains of events. (Poppe et al., 2017b) proposed a method to solve the problem of extracting complete trends of events by cutting the graph into graphlets, and then "stitching" them. This solution has exponential complexity. This approach suffers from both long delays and high memory consumption. The processing time of the algorithm (pattern) affects the system boot because the data is "passed" through the query. The processing time may become unacceptably long. Moreover, with a high intensity of input data flows, the pattern processing link can become a "bottleneck", and the system may be overloaded. In this case, the upcoming events will be lost (if they are not saved) or their processing will be done outside the required screen.

Kolchinsky et al., 2019 proposes CEP optimization methods for processing a stream of events according to several patterns. The optimal plan is built by reordering patterns and sharing them. In this case, these methods cannot be applied, because there is only one Check+ c[] patterns in query Q1.

This article proposes a new method of query implementation, which significantly reduces the time and memory required for event processing. The method is based on the use of a special B-tree based index or a hash table.

2 NOT-COVERED CHECKS FRAUD

Table 1 shows the sequence of events that can occur in the process of circular check kiting using fake checks (Poppe et al., 2017b).

The following particulars are given in Table 1:

- A → B (0) denotes a not-covered deposit event from Bank A into Bank B (check of Bank A is not covered);
- B → A (1) denotes a covered deposit event from Bank B into Bank A (check of Bank A is covered);
- index N means that the check is not covered (Notcovered); for example, 10N means that a check for 10 was issued and it is not yet covered (cash cannot be withdrawn), and 10 (without N) means that the check is covered (cash can be withdrawn).

Table 1 shows that banks have lost (10 + 30 + 20) = 60 units due to fraud.

3 CURRENT FRAUD DETECTION METHOD

To fulfil query Q1, the following method is proposed (Poppe et al., 2017b):

1. When an event e arrives, it is added to the so-called CET graph. For the above example (see Table 1), the CET graph looks like the one in Figure 1. The mark (C) means that the corresponding check is covered.

2. A graph traversal is performed for each event to detect transaction loops. For example, the loop B → C (0), C → B (1) corresponds to the subgraph (C4 → C6). It is also required to define chains of

transactions with payment of not-covered checks in other banks, resulting in the formation of loops.

It is proposed to use one of the algorithms to solve this problem in (Poppe et al., 2017b): the M-CET depth first search (DFS), the T-CET breadth-first search (BFS) or the hybrid H-CET method. The M-CET algorithm is more memory efficient; the T-CET algorithm is more productive in performance and the H-CET algorithm combines the benefits of the two previous methods. The H-CET algorithm assumes cutting the graph into subgraphs (graphlets) (Andreev et al., 2004; Karypis et al., 1995; Tsourakakis et al., 2012). DFS is used for one part of the graphlets and BFS for the others. The problem of cutting a graph into graphlets is a non-trivial task.

The exponential nature of the storage capacity and the number of CPU operations remains for large CET graphs, even for the hybrid H-CET method (Poppe et al., 2017b).

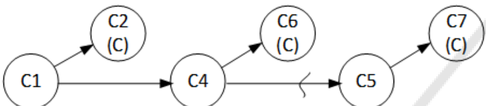

Figure 1: CET graph example.

Table 2 shows an algorithm for executing query Q1 using the Neo4j graph database.

The CET graph is built at the level of transactions between banks in Figure 1. It does not consider that the transaction includes not only the name of the bank, but also the account and check number. A transaction (event) is described by a tuple (source, destination, q) in the developed algorithm. Each of the source and destination elements has three fields (bank, account, check). The feature q has also been introduced: q = 0 - a transaction for paying a check in another bank, e.g., A → B (0), q = 1 - covering a check in another bank, for example, B → A (1).

A node z is created corresponding to the new event e (2 :). Next, a search is for events Y is performed, when the check must be paid or covered in the same bank (and account), which is specified in e.source (3 :). The set Y is equal to {A → B (0), C → B (1)} in the example in Figure 2. Operators 4: -8: establish links between node z and nodes from set Y.

Further along z, a search for a node x is performed with which e forms a loop (9 :). That is, for example, x = A → B (0), z = B → A (1). A loop is a sign of a possible fraudulent transaction. If a loop is found (10 :), search for all chains leading to the beginning of the loop (line 12 :) is in progress. One of them is the loop itself. Figure 2 shows an example of the found chain.

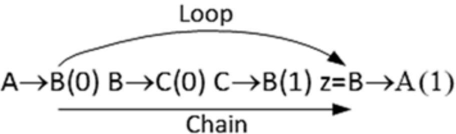

Figure 2: Chain for the loop (A→B(0), B→A(1)).

4 A PROPOSED METHOD FOR DETECTING FRAUD USING AN INDEX

The following features of the flow events can be noted:

1. The right side of the transaction coincides with the left side of the next transaction (NEXT) at the level of the bank name, account number and check, for example, B → C (0) and C → A (0) (see query Q1).

2. A sign of possible fraud is the presence of loops when covering checks, for example, A → B (0) and B → A (1) (see C1, C2 in Table 1).

These features allow you to implement query Q1 using an index (B-tree or hash table). Table 3 shows the flow event algorithm.

Z, y, j are global vars, index1 and index2 are indices (B-tree or hash table) in this algorithm. To speed up the search, event e is stored in the index as two records: (e.source; e.destination, e.q, en1) and (e.destination; e.source, e.q, en1), where the first attribute is the search key (1: , 2 :). The search keys e.source and e.destination may be not unique. The next 4 operators determine if there is a loop, for example, A → B (0) and e = B → A (1) (see Figure 2). First, it is determined whether the check is covered (4 :). Then events A → B (0 or 1) (5 :) are extracted from index 1. And if d (= B) is equal to e.source (= B) and q = 0 (8 :), then there is a loop (9 :). The loop is stored in x, and the strings are searched for leading to the beginning of the loop (10 :). The problem is the same as in Algorithm 1, but it is solved in a different way. The 'chains' procedure is used for this, so let us look at it in details.

The procedure has the following formal input parameters: i is the number of the recursive call level chains, a is the key value for searching in index 2 (source), c1 is the final value in the chain (destination). Let there be a loop A → B (0) and e = B → A (1) (see Figure 2). Then, at the first call to the chains procedure, parameters (1, B, A) (10 :) are

Table 1: Sequence of events that can occur in the process of circular check kiting.

Event	Transaction	Account at Bank A	Account at Bank B	Account at Bank C	Notes
C1	A→B(0)	10_N	(-10)	0	Writing a check for 10_N from Bank A account (initially 0 in A) with the payment from Bank B account (initially 0 in B, hence we see (-10))
C2	B→A(1)	10	(-10) 10_N	0	Writing a check for 10_N from Bank B account to cover the check in Bank A
W3		0	(-10) 10_N	0	Cash withdrawal 10 from Bank A account
C4	B→C(0)	0	(-10) 10_N 20_N	(-20)	Writing a check for 20_N from Bank B account (initially (-10) in B) with the payment in Bank C (initially 0 in C, that's why we see (-20))
C5	C→A(0)	(-30)	(-10) 10_N 20_N	(-20) 30_N	Writing a check for 30_N from Bank C account (initially (-20) in C) with the payment from Bank A account, that's why there is (-30))
C6	C→B(1)	(-30)	(-10) 10_N 20	(-20) 30_N 20_N	Writing a check for 20_N from Bank C account to cover the check in Bank B
C7	A→C(1)	(-30) 30_N	(-10) 10_N 20	(-20) 30 20_N	Writing a check for 30_N from Bank A account to cover the check at Bank C
W8		(-30) 30_N	(-10) 10_N 20	(-20) 20_N	Cash withdrawal 30 from Bank C account
W9		(-30) 30_N	(-10) 10_N	(-20) 20_N	Cash withdrawal 20 from Bank B account

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

Table 2: Algorithm 1 (Graph Database).

Algorithm 1 (Q1)	
Input: e – event	
Output: (x, e) – loop, PATH – chains leading to the beginning of the loop	
1:	i++ // Number of events
2:	z← CREATE(z:Check{id: i, source: e.source, destination: e.destination, q: e.q}) RETURN z
3:	Y ← MATCH (y:Check) WHERE y.destination =e.source RETURN y
4:	IF Y is NOT NULL
5:	FOR n∈ Y
6:	CREATE (n)-[:Next{id1:n.id, id2:z.id}]->(z)
7:	END FOR
8:	END IF
9:	X←MATCH (x:Check) WHERE x.source= z.destination and z.source=x.destination and x.q=0 and z.q=1 RETURN x
10:	IF X is NOT NULL
11:	FOR x∈ X
12:	PATH←MATCH y=(x)-[:Next*]->(z) RETURN relationships(y)
13:	END FOR
14:	END IF

referred to it, and the events A → B (0) and C → B (1) (17 :) will be found. For the first event the loop will be fixed: y [0] [1 ÷ 0] = (A → B (0), B → A (1)) (see 21: -23 :). For the event C → B (1), the chains procedure will be called with parameters (2, C, A) (25 :) and so on. For the event C → B (1), the chains procedure will be called with parameters (2, C, A) (25 :) and so on. Therefore, the chain will be found (see Figure 2): y [2] [3 ÷ 0] = (A → B (0), B → C (0), C → B (1), B → A (1)). Therefore, Algorithm 2 outputs a loop and all possible chains y [][] that contain a loop formed by not-covered check deposits (A → B (0), B → C (0)), as well as fictitious covers (C → B (1), B → A (1)). The loop and chain fall in line with what has been obtained using Algorithm 1.

Operation z [i] ← NULL is necessary because other chains may be shorter. In general, recursive calls to chains form a tree.

According to Algorithm 2, the memory estimate is O (V), and the CPU estimate is O (V) for a hash table and $O(V \cdot \log_K V)$ for a B-index, where K is the number of records in the block index, V is the total number of records. That is less than for Algorithm 1 (Poppe et al., 2017b).

5 EXPERIMENT

Several experiments were conducted to compare the two algorithms. We used a virtual machine running the Ubuntu 20.04.02 x64 operating system based on a 2.3 GHz 8-core Intel Core i9 processor, 8192MB of RAM for the experiments. To test Algorithm 1 we used the Neo4j 4.3.2 graph DBMS, and for Algorithm 2 – PostrgeSQL 12.8. We used BTree (B-Tree Indexes, 2021) and Hash (Re-Introducing Hash Indexes in PostgreSQL, 2021) indexes.

The source data was generated with the following components:

- Source Bank - the bank from the left side of the transaction;
- Source Account - account number from the left side of the transaction;
- Source Check - check number from the left side of the transaction;
- Destination Bank - the bank from the right side of the transaction;
- Destination Account - account number from the right side of the transaction;
- Destination Check - check number from the right side of the transaction;
- Date and time of the transaction;
- Direction of the transaction.

Testing was conducted with 10 transactions per second for 1 day. Loops appeared every 30 minutes. A smaller dataset was used for Algorithm 1 due to the large processing time in the Neo4j DBMS. That is, if Algorithm 2 processes about 900,000 events, then Algorithm 1 supports only 100,000 events.

Under the experiments three main indicators were measured: Instructions per Cycle (IPC), the amount of RAM and the processing time for one event. The load on the CPU for Algorithm 2 is much lower than for Algorithm 1 (Figure 3, Figure 4). The graph in Figure 4 shows that the processor is overloaded, and some processes are waiting (when the load is greater than 1).

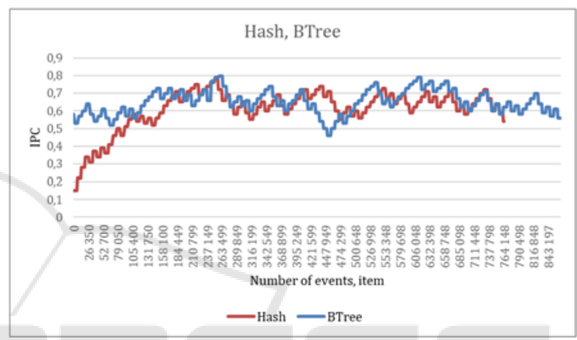

Figure 3: IPC, Algorithm 2.

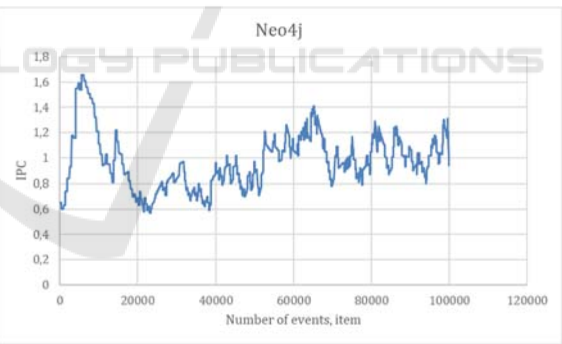

Figure 4: IPC, Algorithm 1.

The storage capacity is not the most typical indicator in these tests due to the peculiarities of Neo4j (Figure 5, Figure 6). It periodically frees memory as shown in Figure 6. This is due to «garbage collection».

The processing time for one event using Algorithm 2 is short because the test case is stored in the RAM and the processing is carried out in the CPU (Figure 7 and 8). In fact, events are selected from the stream and the processing time will be longer there. It is not the absolute values that are important in this study, but the ratio of the processing time of one event using Algorithms 1 and 2. The average processing

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

time using Algorithm 2 is almost 37 / 0.66 = 56 times less with the number of events equal to 100,000.

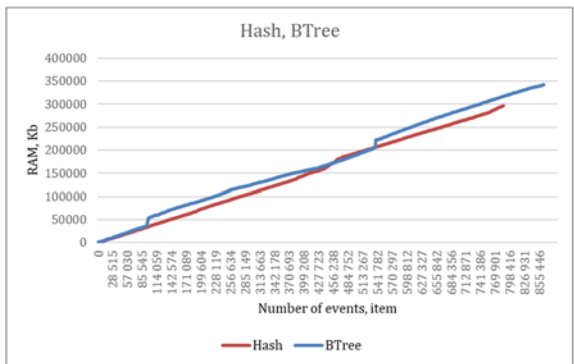

Figure 5: Volume of RAM, Algorithm 2.

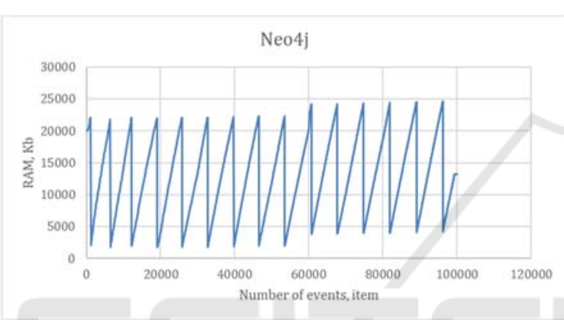

Figure 6: Volume of RAM, Algorithm 1.

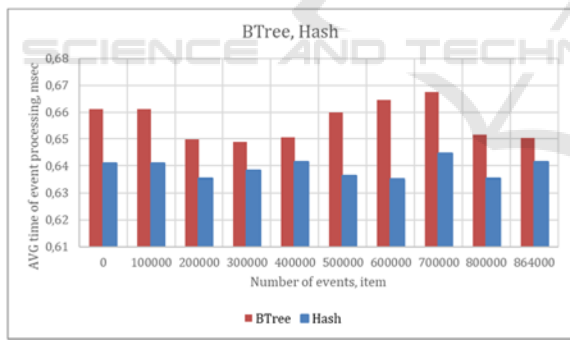

Figure 7: One event average processing time, Algorithm 2.

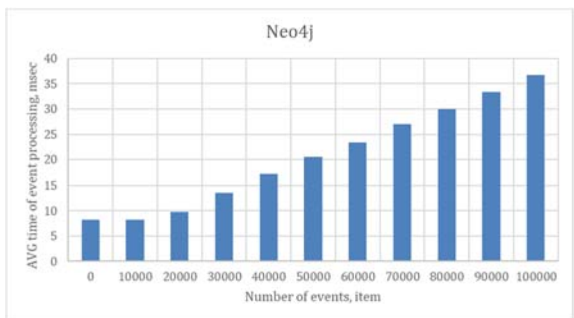

Figure 8: One event average processing time, Algorithm 1.

The maximum time for Algorithm 2 fluctuates in the range of 20-40 msec with a smooth mode of vibration except for surges up to 100 msec (Figure 9). The maximum time for Algorithm 1 fluctuates in the range of 40-90 ms with a stick-slip nature mode of vibration (Figure 10).

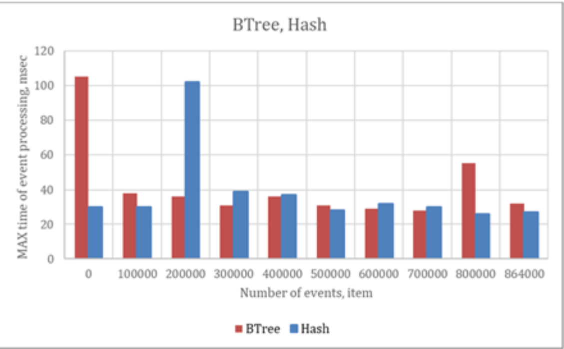

Figure 9: Maximum processing time for one event, Algorithm 2.

A sliding window was added to Algorithms 1 and 2. No significant changes are observed for Algorithm 2 (Figure 11). Figure 12 shows that the processing time for one event begins to increase much more slowly with a sliding window for Algorithm 1.

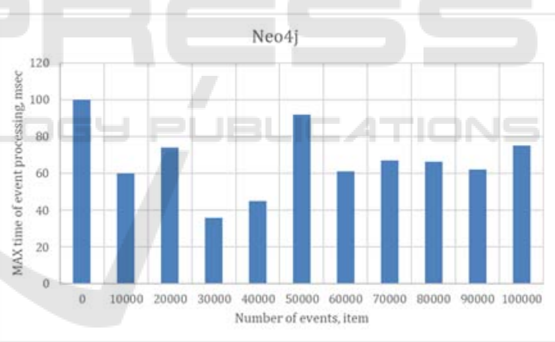

Figure 10: Maximum processing time for one event, Algorithm 1.

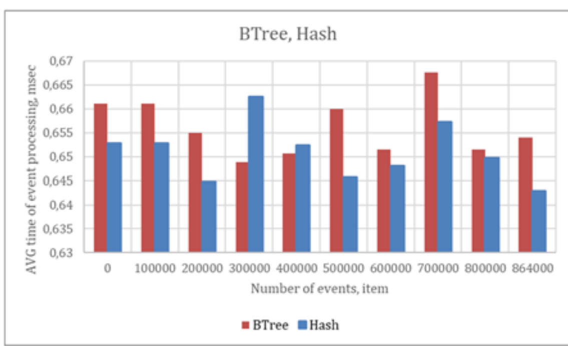

Figure 11: Average processing time for one event with a sliding window, Algorithm 2.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

Table 3: Algorithm 2 (Relational Database).

Algorithm 2 (Q1)	
Input: e – event, en1 – event number	
Output: (x, e) – loop, y[][] – chains leading to the beginning of the loop	
1:	index1.add(s=e.source; d=e.destination, q=e.q, n=en1) // key; value – for loops search
2:	index2.add(d=e.destination; s=e.source, q=e.q, n=en1) // key; value – for chains search
3:	z[0] ← (s=e.source, d=e.destination, q=e.q, n=en1)
4:	IF e.q==1 // check coverage?
5:	D←index1.search(s=e.destination) // d and q saved into D
6:	IF D is NOT NULL
7:	FOR (d,q) in D
8:	IF d==e.source and q==0
9:	x←(e.destination, d) // loop
10:	chains(1, e.source, e.destination) // searching for chains
11:	END IF
12:	END FOR
13:	END IF
14:	END IF
15:	Return
16:	chains(i, a, c1)
17:	B←index2.search(d=a) // (s,d, q, n) values are saved into B
18:	i1←i+1
19:	FOR (s,d,q,n) in B && n<z[i-1].n // events in one chain are unique
20:	z[i]← (s, d, q, n)
21:	IF s==c1 && d==z[0].s && q==0
22:	y[j++][]←z[];
23:	z[i]←NULL
24:	ELSE
25:	chains(i1, s, c1)
26:	END IF
27:	END FOR
28:	z[i]←NULL
29:	Return

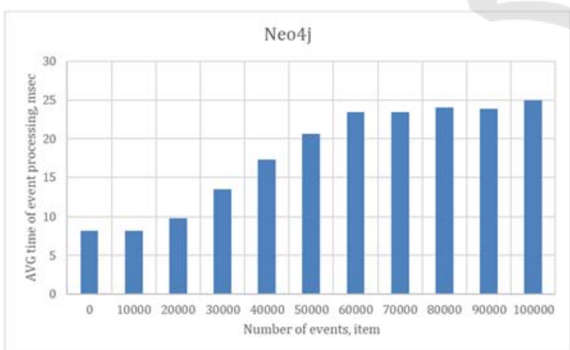

Figure 12: Average processing time for one event with a sliding window, Algorithm 1.

6 CONCLUSIONS

This article offers a new way to detect not-covered bank check fraud. This is one of the most difficult types of fraud to detect due to the complexity of retrieving complete chains of check transactions. This problem is solved using Complex Event Processing technology (CEP). It is used in tasks of processing many events in real time.

The proposed Algorithm 2 based on indexing methods is superior to the existing Algorithm 1. It demonstrates better results for the following three main indicators of the existing Algorithm 1 based on graphs: IPC, the amount of RAM occupied and the processing time of one event.

The considered approach to the analysis of event trends is planned to be used for a wider class of queries as a part of the future work.

REFERENCES

Agrawal, J. et al. (2008). Efficient pattern matching over event streams. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*.

Andreev, K., Racke, H. (2004). Balanced graph partitioning. In *SPAA- 2004*.
B-Tree Indexes. (2021). https://www.postgresql.org/docs/11/btree.html
Karypis, G., Kumar, V. (1995). Multilevel graph partitioning schemes. In *Parallel Processing*.
Kolchinsky, I., Schuster, A. (2019). Real-time multi-pattern detection over event streams. In *Proceedings of the 2019 International Conference on Management of Data*.
Luckham, D. C. (2011). Event processing for business: organizing the real-time enterprise. In *John Wiley & Sons*.
Ma, L., Lei, C., Poppe, O., Rundensteiner, E. A. (2022). Gloria: Graph-based Sharing Optimizer for Event Trend Aggregation. In *Proceedings of the 2022 International Conference on Management of Data.*
Piyush, Yadav, Edward, Curry (2019). VidCEP: Complex Event Processing Framework to Detect Spatiotemporal Patterns in Video Streams. In *2019 IEEE International Conference on Big Data (Big Data).*
Poppe, O., Lei, C., Rundensteiner, E. A., Maier, D. (2017a). GRETA: Graph-based Real-time Event Trend Aggregation. *Proceedings of the VLDB Endowment*, *11*(1).
Poppe, O., Lei, C., Ahmed, S., Rundensteiner, E. A. (2017b). Complete event trend detection in high-rate event streams. In *Proceedings of the 2017 ACM International Conference on Management of Data.*
Poppe, O. et al. (2019). Event Trend Aggregation Under Rich Event Matching Semantics. In *Proceedings of the 2019 International Conference on Management of Data*.
Re-Introducing Hash Indexes in PostgreSQL (2021). https://hakibenita.com/postgresql-hash-index
The Press Enterprise (2015). http://www.pe.com/articles/checks-694614-people-bank.html
Tsourakakis, C. E. et al. (2012). Streaming graph partitioning for massive scale graphs. In *Technical report*.
Volkova, M.M., Antonova, P.V., Shameeva, A.R. (2020). High-Performance Complex Event Processing. In *2020 International Multi-Conference on Industrial Engineering and Modern Technologies (FarEastCon).*
Wu, E., Diao, Y., Rizvi, S. (2006). High-performance complex event processing over streams. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data.*
Zhang, H., Diao, Y., Immerman, N. (2014). On complexity and optimization of expensive queries in complex event processing. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data.*

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS