A Study about FP-growth on a Distributed System Using Homomorphic Encryption

Mayuko Tanemura

Master School of Computer Science
Ochanomizu University
Tokyo, Japan
Email: mtanemura@is.ocha.ac.jp

Masato Oguchi

Ochanomizu University
Tokyo, Japan
Email: oguchi@is.ocha.ac.jp

***Abstract*—Large data collections, such as big data, are utilized and analyzed in business.Because large-scale data calculations require a computer system with high processing power, it is practical to outsource the processing to an external server. However, especially when consigning confidential data, such as personal information, it is important to take measures against information leakage. There are various methods, such as data anonymization processing for privacy protection, but in this research, as a method of data confidentiality protection, a Fully Homomorphic Encryption (FHE) that can be calculated in an encrypted state is used. As in a previous study, P3CC (Privacy Preserving Protocol for Counting Candidates) is used, which applies FHE to a client-server type system that performs frequent pattern mining with the Apriori algorithm. To implement this system, the Apriori algorithm is changed to the FP-growth (frequent pattern growth) algorithm in our research work, and the results are compared with those of the existing method using the Apriori algorithm.**

Keywords*–*Fully Homomorphic Encryption; Data Mining; Distributed System.

I. INTRODUCTION

Big data and large-scale data have been utilized in various fields of business. To perform calculations that deal with large-size data, a computer system with a high processing power is required. When it is difficult to provide such a system, external computing resources, such as the cloud can be used. In that case, it is necessary to take sufficient measures to prevent the leakage of the information to be transmitted during the consignment processing of data, such as personal information and medical data, for which it is necessary to ensure confidentiality. When the processing of confidential information is outsourced, it is possible to perform addition and multiplication calculation processing without showing plaintext data to the consignee server by using fully homomorphic encryption (FHE). Therefore, there is an expectation that data can be entrusted even if transmitted by a server that is not reliable. As an application example of FHE, Liu et al.'s P3CC [1] performed the frequent pattern mining of transaction data by the Apriori algorithm; studies on speeding up the approach have also been conducted [2] [3], in addition to distributed processing using the FUP (Fast Update) algorithm [4], as described in Section IV. In this research, we implement the system of frequent pattern mining but with the part processed by the FP-growth algorithm instead of by Apriori, and we compare the results with those of previous studies. The remainder of this paper is organized as follows. Sections II and III introduce related technologies. In Section IV, some of the previous researches are shown. Sections VI and VII-A are about the experiment conducted in this research. Finally, in Section VIII, the conclusion and future plans are stated.

II. FULLY HOMOMORPHIC ENCRYPTION

A. Overview

Privacy homomorphism was firstly proposed by Rivest et al. [5]. They proposed privacy homomorphism as the property of being able to perform operations while being encrypted. FHE is an encryption method that combines the features of additive homomorphism and multiplicative homomorphism.

This approach has the function of a public key cryptosystem, and capabilities for the addition and multiplication of ciphertexts in an encrypted state are established. In other words, it is possible to manipulate the plaintext before it is decrypted by computing the ciphertexts. Therefore, by using FHE, there is an expectation that calculation processing can be outsourced without showing plaintext data to the outsourced server.

For FHE, Gentry proposed a lattice-based implementation method in 2009 [6]. In each ciphertext, random noise is added to increase the indecipherability of the encryption.

The problem is that the computational complexity of the process can be massive because of the large data size of the ciphertext and key, and the value of the noise increases each time the ciphertext is calculated. If the noise exceeds the threshold, the decryption becomes impossible. The noise value increases significantly, especially when performing multiplication. By performing a bootstrapping process, it is possible to refresh the noise of the ciphertext, but this process also requires a large amount of calculation.

B. Leveled FHE

Leveled FHE is an implementation of FHE that does not use bootstrapping, and it was proposed by Brakerski et al. [7]. Leveled FHE is a structure of perfect homomorphic cryptography that can evaluate the result of a logic circuit of fixed depth L. If the calculation logic circuit depth is small enough for the level given in advance, there is no need to do bootstrapping. This study uses Leveled FHE, so it does not use bootstrapping.

III. Frequent Pattern Mining

Frequent pattern mining is a method aimed at extracting correlation rules from a large quantity of data. This study deals with transaction data.

In the frequent pattern mining used in this research, which item is included in each transaction is represented as a binary matrix. Frequent item sets are extracted based on whether the support value is greater than or equal to the given minimum support value. The support value of each item set is the ratio of the number of transactions including the item set to the total number of transactions. Apriori and FP-growth are typical algorithms for frequent pattern mining. The outline of each algorithm is provided below.

A. Apriori

Apriori algorithm was proposed by Agrawal et al. in 1993 [8]. Apriori is a breadth-first search algorithm that compares the support value III with the given minimum support value in order to acquire the list of frequent item set from size of 1. Apriori is used in the previous research described in Section IV. Even if there are few types of items, the number of possible combinations of items can be massive. To reduce the calculation amount, pruning is adopted. For example, if the support value of an item set of item length n is less than the minimum support value, it is determined that the pattern of item length $n + 1$ including the item set is not frequent anymore. The implementation of Apriori is relatively easy.

B. FP-growth

FP-growth is another algorithm used to extract frequent item sets. In contrast to Apriori, FP-growth is a depth-first search. The results of these two algorithms, the list of frequent itemsets, will correspond to each other. The process of FP-growth is different from Apriori in terms of the data structure; it uses tree-structured data to search for frequent itemsets. First, the database is scanned, and the transaction data are stored in the tree structure of a prefix tree called FP-tree. Then, frequent patterns are found by scanning the tree. It is different from Apriori in that it does not enumerate the frequent item set candidates. Depending on the data size and data characteristics, there is an expectation that the search can be made more efficient than Apriori, in which the enumeration of frequent items becomes a bottleneck. The implementation is also more complex than that of Apriori.
FP-tree is defined as follows [9]:

1) It consists of one root labeled as ”null”, a set of item prefix subtrees as the children of the root, and a frequent-item header table.
2) Each node in the item prefix subtree consists of three fields: item-name, count, and nodelink, where item-name registers which item this node represents, count registers the number of transactions represented by the portion of the path reaching this node, and node-link links to the next node in the FP-tree carrying the same item-name, or null if there is none.
3) Each entry in the frequent-item header table consists of two fields, (1) item-name and (2) head of node-link, which points to the first node in the FP-tree carrying the item-name.

To construct an FP-tree, transaction data and minimum support values are needed as input data. To pick the frequent item sets, scan the constructed FP-tree recursively.

IV. Previous Research

An outline of the previous research on secure data mining using FHE is provided below. The algorithm for frequent pattern mining adopted in all of the previous research mentioned in this section is Apriori.

A. P3CC

P3CC is a secure method for a frequent pattern mining consignment system using FHE proposed by Liu et al. [1]. Comparing operations between ciphertexts are difficult when FHE is used. Therefore, when the values of ciphertexts need to be compared, they are sent back to the client machine. To reduce the data size of the ciphertext, the numbers of items and transactions are not encrypted, and only a binary matrix that represents which items each transaction includes is encrypted. Furthermore, adding dummy data on the client side prevents the guessing of plaintext data from the server.

B. Speedup of P3CC with Cipher Text Packing and Cipher Text Caching

Imabayashi et al. [2] proposed a method to speed up P3CC by introducing a packing scheme by Smart and Vercauteren [10]. The method reduced the amount of ciphertext and the multiplication of ciphertexts using ciphertext packing by encrypting multiple integers as a vector. As a result, the process achieved a 10-fold speedup compared to the case without packing. This method can be applied not only to Apriori, but also to secure search and other data mining algorithms. Then, they proposed a method of caching a ciphertext to reduce the space-time complexity of frequent pattern mining with FHE. It is shown that the proposed method can greatly reduce the execution time and memory usage of Apriori by P3CC, and the effect is larger when the data set is large or when a dummy set is added. In particular, when the number of transactions is 10,000, a 430-fold faster speed and 94.7 % memory usage reduction are realized compared to P3CC [3].

C. Implementation of P3CC in a distributed environment and speeding up update with FUP algorithm

Yamamoto et al. [4] implemented a secure data mining system using the FUP algorithm to speed up Apriori algorithm when the database is updated [4]. They also applied a master/worker-type distributed processing to speed up the system. Item set division was applied to the distributed processing method. As a result, the execution time of the recalculation when the FUP algorithm is introduced at the time of the database update can be shortened by approximately 3- to 4-fold compared with recalculation by the Apriori algorithm. Additionally, due to the decentralized processing, the calculation time on the master side is reduced according to the number of distributed machines.

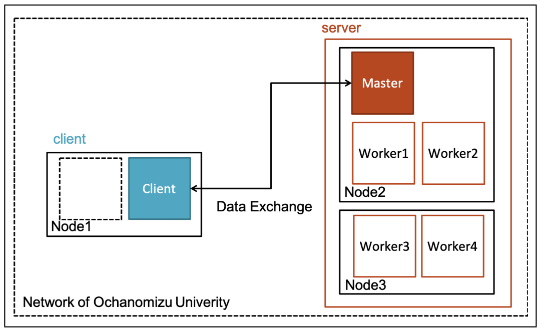


Figure 1. execution time when FP-growth was used (Client)

V. Implementation of Secure Frequent Pattern Mining

In this research, a system like the one presented Figure 1 is used. In this study, a master-worker (master-slave) distributed system is adopted for processing on the server. The number of workers is set from 1 to 4. Two algorithms of frequent pattern mining are used in this system: FP-growth and Apriori (from previous research). For both programs, we used ciphertext packing based on previous research [2] by Imabayashi et al. [2]. The program using Apriori is the one produced by Yamamoto et al. [4]. This system was implemented in C++, the library of the FHE is Helib [11], and the distributed/parallel processing library is MPI [12].

The following shows the processing for each system using the two types of algorithms.

A. *Outline of processing using Apriori*

The process is partially delegated to the server, as indicated in Section IV. The procedure is shown below.

1) Preparing data on the client
   a) Encrypt transaction data with FHE.
   b) Create candidate items and send them to the server.
2) Consigned processing on the server
   a) Receive encrypted data from client and calculate support value of item without decryption. Then, send the result back to the client. In this process, master-worker type distributed processing is performed.
3) Comparison with minimum support value on the client
   a) Receive file from the server
   b) Retrieve items whose support value is equal to or greater than the minimum support value.
   c) Return to procedure 1b and send the candidate whose item set size is one larger. Repeat until the number of candidates is 0.

B. *Outline of processing using FP-growth*

When using FP-growth, the procedure is similar to the process with Apriori in the first step. In this program, the calculation of the support value of the item, which is the first step of the construction of FP-tree, is consigned to the server, while the construction and scanning of the FP-tree are done on the client machine. Because it is necessary to perform many comparison operations to construct and scan an FP-tree, in this research, these processes were performed on the client. Even if the process on the client is not finished, the server ends the program after sending the support value calculation result to the client. The procedure FP-tree construction and scanning is shown below.

1) Preparing data on the client
   a) Encrypt transaction data with FHE.
   b) Create candidate items and send them to the server.
2) Consigned processing on the server
   a) Receive encrypted data from client and calculate support value of item without decryption. Then, send the result back to the client. In this process, master-worker type distributed processing is performed.
3) FP-tree construction on client
   a) Receive file from the server.
   b) Retrieve items whose support value is equal to or greater than the minimum support value.
   c) Sort the items in the order of occurrence and recreate the transaction excluding the items that are not frequent.
   d) Construct an FP-tree.
4) FP-tree scanning on the client
   a) Scan the constructed FP-tree and output the result.

VI. Experiment

A. *Experiment outline*

The client program and server program were run on each computer using two computers in the same network. We confirm the execution time by changing the minimum support value and the number of transactions.

B. *Experiment environment*

The performance of the computer used in the experiment is shown in Table I.

TABLE I. DETAILS OF THE MACHINE USED IN THE EXPERIMENT

OS	CentOS 6.9
CPU	Intel® Xeon® Processor E5-2643 v3 3.6 GHz 6 core 12 threads
Memory	512 GB

One machine of the type shown in the Table I was used as a client and a server. On the server side, as shown in Section V, master-worker type distributed processing is performed, but in this experiment, it is fixed to one. Additionally, the worker runs on the same machine as the master.

C. *Experiment method*

1) Investigation of the lowest possible level for experimental data: For each transaction data, when minimum support value is 0.01, an experiment was conducted with the possible lowest level. In this case, the lowest level represents the lowest one that the cyphertext can be normally decoded until the end of the calculation. The level was specified by trial. In all experiments in this study, the level is set to 17 in Apriori and 3 in FP-growth. These values of the level are the lowest that can be applied to each method. If the levels are lower than these values, execution will be end with a decode error.

2) Measurement of Execution time: The execution time was measured for each combination of input data and minimum support value.

3) Measurement of Resource Usage: CPU and memory usage are measured by dstat command.

D. Input data

The input data in this experiment were artificially created. Data generation was performed using IBM Quest Synthetic Data Generator, and the parameters were generated by specifying the average item length, the maximum pattern size, the number of item types, and the number of transactions.

TABLE II. PARAMETERS OF INPUT DATA

Average transaction length	5
Maximum pattern size	5
Number of item types	30
Number of transactions	9900

The value of each parameter is specified as shown in Table II.

VII. RESULTS AND DISCUSSION

A. Results

The execution time shown is the average of three trials for each parameter. Figure 2 shows the execution time on the client when FP-growth is used.

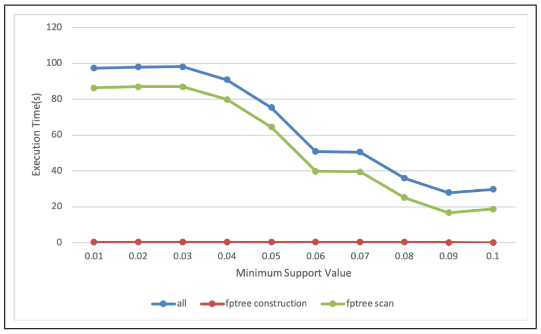


Figure 2. execution time when FP-growth was used (Client)

The execution time on the client side decreases almost monotonically as the support value increases. It is the FP-tree scan that is most affected by the computation time, and this changes with the size of the tree created. The creation time of the tree itself ends in linear time, but the scanning requires recursive processing, so when the minimum support value is small, the calculation time tends to jump. In the current implementation, the calculation time on the server side is approximately 10 seconds and hardly changes because the process is the same if the data are the same.

Second, the execution time when FP-growth was distributed is shown in Figure 3.

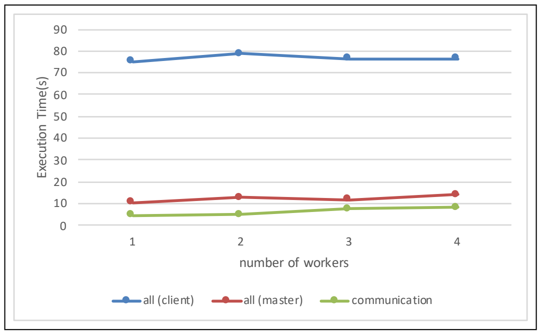
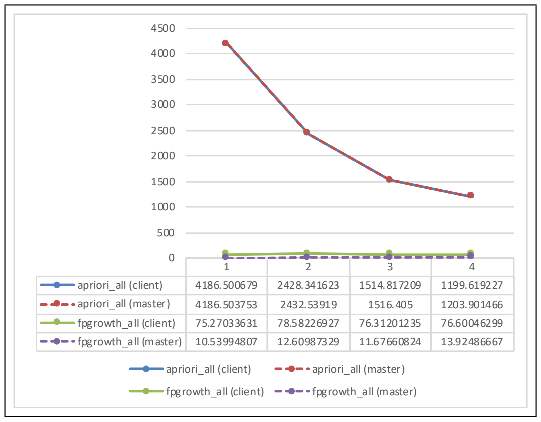


Figure 3. execution time when FP-growth was used (distributed)

The communication time increases as the number of workers increases. However, since there were few parts of the processing that were entrusted to the servers, the overhead was large and the distribution effect was not so great.

A comparison between two programs with the two different algorithms is shown in Figure 4.



	1	2	3	4
apriori_all (client)	4186.500679	2428.341623	1514.817209	1199.619227
apriori_all (master)	4186.503753	2432.53919	1516.405	1203.901466
fpgrowth_all (client)	75.27033631	78.58226927	76.31201235	76.60046299
fpgrowth_all (master)	10.53994807	12.60987329	11.67660824	13.92486667

Figure 4. comparison of execution time between the programs using Apriori and FP-growth

The minimum value of the required level differs significantly between the systems using the two algorithms. This difference directly affects the size of the ciphertext and the execution time.

Figure 5 and Figure 6 show a comparison of the resources used by the client's machine. The data is measured when it is run with four workers and minimum support value is 0.05. In the system by Apriori, the CPU utilization peaked at the timing of receiving data. In FP-growth, after receiving data, FP-tree scan is performed in the part where the value is continuously 10%. In addition, the memory usage rate continued to be high during entrustment processing in both programs.

B. Discussion

When Apriori is used, once the transaction data are encrypted, re-decryption is not performed every time when data is exchanged with the server. Therefore, as the number of loops of the calculation increases, in other words, as the maximum value of the frequent item set length increases, the amount of noise in the ciphertext increases. To prevent decryption error, it

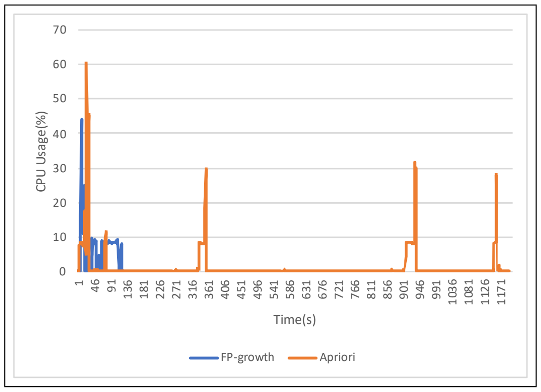


Figure 5. comparison of cpu usage between the programs using Apriori and FP-growth

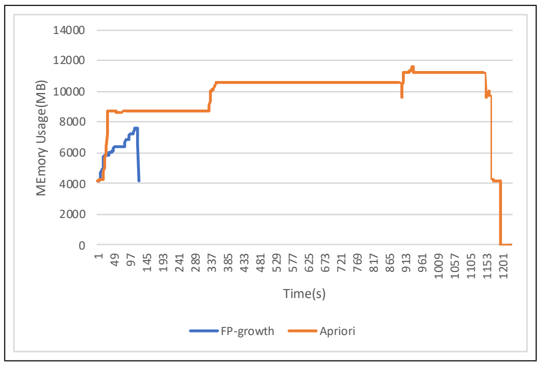


Figure 6. comparison of memory usage amount between the programs using Apriori and FP-growth

is necessary to set the value of the ciphertext level sufficiently large in advance.

On the other hand, in this implemetation, transaction data exchange with the server is performed only once in FP-growth; the initial value of the level needs to be able to calculate the support value only one time, so the required level compared with Apriori is smaller.

In addition, it can be seen that the increase in the number of candidate item sets more strongly affects the required level than the number of transactions due to the change in the support value. It can be confirmed that the execution time of FP-growth is approximately 100-fold smaller than the execution time of Apriori at maximum. The execution time is longer in Apriori's system. The higher the level is, the larger the size of the ciphertext, and the amount of processing could also be large. In FP-growth, although the amount of computation in scanning of the FP-tree is also large, it was found that it is extremely small when compared with the processing of the ciphertext in this experiment.

In the experiment, it is confirmed that the difference in the results between the two different systems was not only due to the difference in the algorithm itself but the difference in implementation also had a relatively large effect. In the current implementation using FP-growth, although the system using Apriori exchanges with the server multiple times, the portion of the processing entrusted to the server is small. The amount of the process dealing with cyphertext is generally large, so when the number of processes used to manipulate the ciphertext is reduced, a large difference is observed in the execution time.

VIII. Conclusion and Future Plans

A system doing frequent pattern mining by the FP-growth algorithm using a FHE was implemented. Then, the execution time and the amount of resource usage of this system were measured, and they were compared with previous system using the Apriori algorithm. In the comparison of the execution time, the system using FP-growth was approximately 100-fold faster than the system of the previous research. This result is considered to be the reason for much of the difference between the implementation of the system, rather than the algorithm itself. In the future, to improve the FP-growth system, it is considered necessary to reduce the number of times of the transmission and reception of the ciphertext data between the client and the server when increasing the ratio of processing on the server.

Acknowledgment

This research could not have been done without considerable support and advice from Professor Hayato Yamana as well as the professors and other researchers who belong to the research group. This research is supported by JST CREST JPMJCR1503.

References

[1] J. Liu, J. Li, S. Xu, and B. C. Fung, "Secure outsourced frequent pattern mining by fully homomorphic encryption," in International Conference on Big Data Analytics and Knowledge Discovery. Springer, 2015, pp. 70–81.

[2] H. Imabayashi, Y. Ishimaki, A. Umayabara, H. Sato, and H. Yamana, "Secure Frequent Pattern Mining by Fully Homomorphic Encryption with Ciphertext Packing," in Data Privacy Management and Security Assurance, G. Livraga, V. Torra, A. Aldini, F. Martinelli, and N. Suri, Eds. Cham: Springer International Publishing, 2016, pp. 181–195.

[3] ——, "Streamline Computation of Secure Frequent Pattern Mining by Fully Homomorphic Encryption," IPSJ TOD, vol. 10, no. 1, mar 2017, pp. 1–12.

[4] Y. Yamamoto and M. Oguchi, "Distributed System for Secret Data Mining while Updating Large Itemsets using Fully Homomorphic Encryption," in Multimedia, Distributed, Cooperative, and Mobile Symposium, vol. 2018, jun 2018, pp. 992–998.

[5] R. L. Rivest et al., "On data banks and privacy homomorphisms," Foundations of secure computation, vol. 4, no. 11, 1978, pp. 169–180.

[6] C. Gentry, "A Fully Homomorphic Encryption Scheme," Ph.D. dissertation, Stanford, CA, USA, 2009, aAI3382729.

[7] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) Fully Homomorphic Encryption Without Bootstrapping," ACM Trans. Comput. Theory, vol. 6, no. 3, Jul. 2014, pp. 13:1–13:36. [Online]. Available: http://doi.acm.org/10.1145/2633600

[8] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in Acm sigmod record, vol. 22, no. 2. ACM, 1993, pp. 207–216.

[9] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns Without Candidate Generation," in Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, ser. SIGMOD '00. New York, NY, USA: ACM, 2000, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/342009.335372

[10] N. P. Smart and F. Vercauteren, "Fully homomorphic SIMD operations," Designs, codes and cryptography, vol. 71, no. 1, 2014, pp. 57–81.

[11] HElib. https://github.com/homenc/HElib (visited on 18/09/2019).

[12] Open MPI. https://www.open-mpi.org/ (visited on 18/09/2019).