International Journal of u- and e- Service, Science and Technology
Vol. 5, No. 2, June, 2012

An Efficient RFID Data Processing Scheme for Data Filtering and Recognition

Sih-Ying Li[1], Hsu-Yang Kung[1], Chi-Hua Chen[2,*] and Wen-Hsi Lydia Hsu[3]

*[1]Department of Management Information Systems,
National Pingtung University of Science and Technology, Pingtung, 912 Taiwan
[2]Institute of Information Management,
National Chiao-Tung University, Hsinchu, 300 Taiwan
[3]Department of Business Administration,
National Pingtung University of Science and Technology, Pingtung, 912 Taiwan
{M9956006, kung, hsuw}@mail.npust.edu.tw, [*]chihua0826@gmail.com*

Abstract

Radio frequency identification (RFID) technology has been toward a mature stage. Many industries build the RFID technology into the production process. The RFID technologies can support the firms in the supply chains to share information and increase the achieved transparency of process control and complete traceability records. At present, more RFID applications have been extended to life by the logistics management application. RFID developments of the future are a large-scale deployment, and information sharing with each other to achieve the convenience of multiple applications. To achieve the above purpose, the core function of RFID middleware is to achieve effective data collection and information sharing. Since RFID can generate large amounts of data flow in a short time, RFID middleware has to fast filter duplicated data and identify valid data. In this work, we design a data processing method to achieve on effective data filtering control. We design the filtering method and the effective hash functions which use the EPC coding of characteristics combining tree structure. The proposed middleware can reduce large amounts of data from readers by filtering duplicated data. Finally, we simulated the proposed data processing control schemes and revealed the good performance results.

Keywords: *RFID data filtering, RFID data identification, Hash function, EPC*

1. Introduction

In recent years, *radio frequency identification* (RFID) technology has been widely as a policy to promote technology projects. RFID technology can be used to forecast a significant change in consumption habits and human live.

RFID can generate large amounts of data in a short time, and RFID middleware can collect and integrate these data [10, 3, 2, 7, 8]. Therefore, this study expects to develop an effective method of RFID data processing that can help RFID middleware quickly and effectively deal with large numbers of RFID data.

This study uses the *electronic product code* (EPC) [5] to construct the tree structure, and design a hash function to reduce the data length for hash value generation. The hash value will be a basis filtering code.

The remainder of this work is organized as follows. In section 2, we introduce related work on RFID middleware. Section 3 shows the architecture of RFID middleware. In section 4, we present the hash function to filter RFID data. In section 5, we show the system implementation. In section 6, we use simulations to improve essential efficiency of the method. Finally, section 7 concludes this work.

2. Related Work

The middleware is divided into four methods which are proposed in previous work.

2.1. Device Adaption and Management

Liu et al. [9] proposed RFID middleware device management based on agents. These agents were the control agent and device agents. The core component is the control agent which manages all device agents and their lifecycles.

2.2. Data Processing

Anagnostopoulos et al. [1] designed a filter that receives input from readers or other filters. The filter input includes the distributed parameters of reference services (e.g., database queries and web services). The filtering process can be defined as a set of XML documents.

Hsiao [6] proposed a filtering strategy using bit information in RFID. His strategy is based on the EPC code and RFID tag which is encoded by the position of the order and converted into binary code. Moreover, the bit position record method can support re-storage, processing and up to speed for filtering.

2.3. Process Design

Anagnostopoulos et al. [1] proposed a filtering process from data input to output which is defined by a XML file. Additionally, a filter can generate more than one output for a user with the situation on the application.

2.4. Efficiency

EPCglobal proposed the distributed middleware architecture, application level events (ALE). For example, Mo et al. [11] presented project of RFID application. There are including device control, data filtering, and storage to companies as a group, alone the burden of the workload. Cui et al. [4] developed the method of middleware load balancing. This method constructs multi-host middleware and shares information to other middleware. When the workload exceeds a middleware threshold, some readers will be transferred to another middleware.

3. Systematic Structure

The RFID middleware architecture which comprises the RFID devices, middleware, and application is shown in Figure 1. The structure of these components is described as follows.

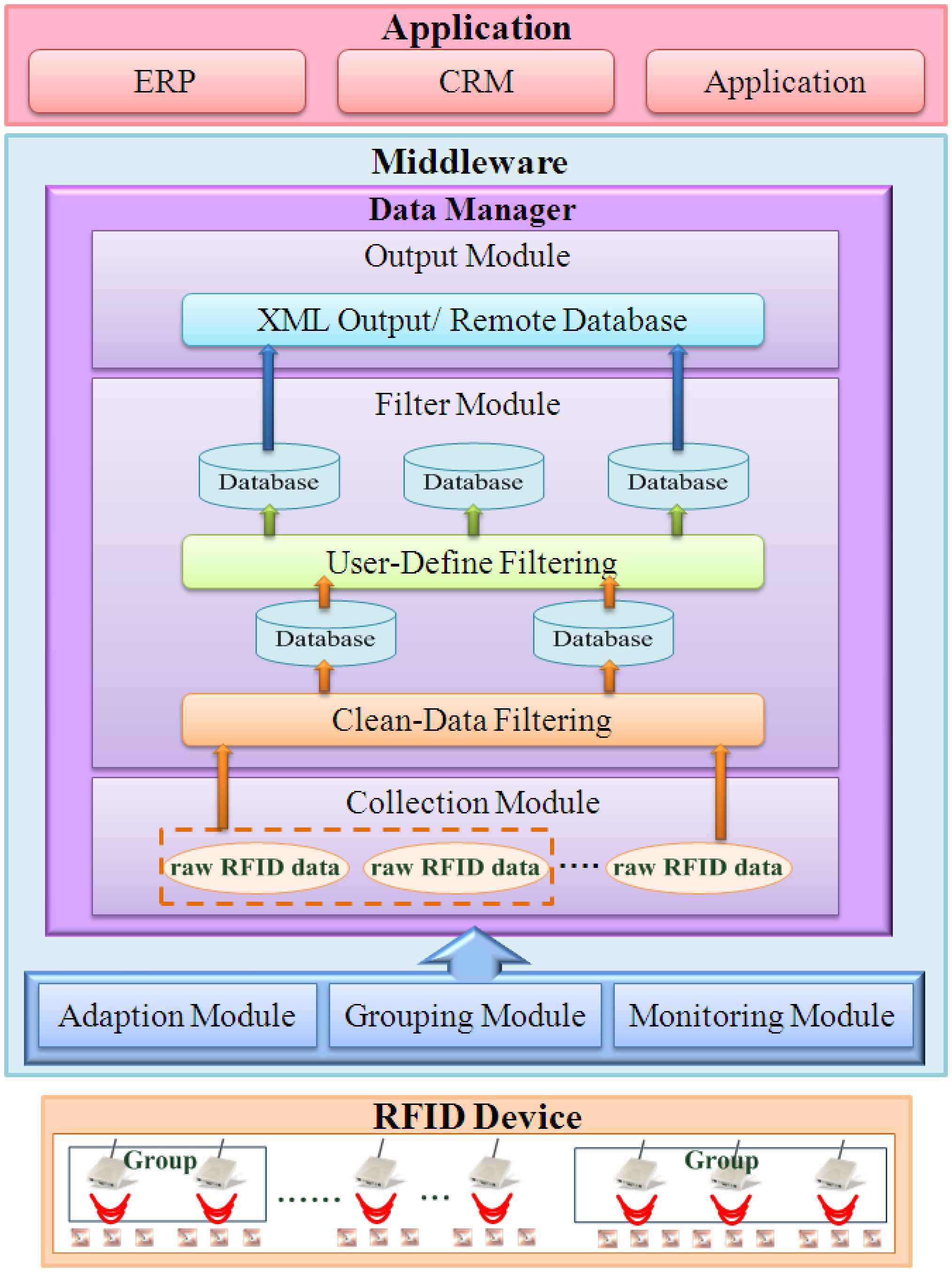


Figure 1. RFID Middleware Architecture

3.1. RFID Device

This part describes the RFID readers and tags which have been designed by the different equipment specifications. Middleware must connect various readers to achieve the aim of a single user interface.

3.2. Middleware

Middleware has two core components which are the data manager and reader controller. The data manager is responsible for data filtering, collection, and storage. The reader controller is responsible for connecting readers, and grouping and monitoring readers.

3.2.1. Data Manager

3.2.1.1. Collection Module: The reader effectively collects and transmits data to the corresponding filtration module for processing.

3.2.1.2. Filter Module: This module has multiple filter functions. This module uses the hash function to filter duplicate data, and lets users filter specific data. That function is also called data classification.

3.2.1.3. Output Module: A user can set a data flow as output from the filter module. Additionally, data can be saved to a database specified by a user or as output data in the XML format.

3.2.2. Reader Controller

3.2.2.1. Monitoring Module: This module can support users to manage readers. The users can monitor the working condition of readers for real-time monitoring and troubleshooting.

3.2.2.2. Grouping Module: This module manages many readers which are used for different application scenarios. Thus, this work designs a grouping function, and the readers which are in a group are used for the same application scenarios. A user can then control multiple readers in a group simultaneously.

3.2.2.3. Adaption Module: This module controls readers from different manufacturers for users.

3.3. Application

The data was transformed to information by middleware. Then this module delivers information to other application system or database. Therefore, a user can refer to the designation database or obtain data in the XML format.

4. Evaluation

4.1. Clean-data Filtering

Clean-data filtering filters duplicate data to reduce the amount of data. This work sets a particular unit of read the range. It is not a reader of a read the range. Therefore, it is the same between the same readers reads tags and the time difference in repeated and overlapping multi-readers. Traditional methods repeat data filtering, such that a reader reads a new tag, the reader have to read into the previously all tag data will be compared all tags to determine whether data is repeated. However, the approach requires a considerable amount of resources and computes in the slow system. To increase the filtration rate, this work uses a hash function for repeat information filtering. We use the hash function to design filter made of two methods: (1) hash function and (2) tree structure with the hash function.

4.1.1. Hash Function

We propose the method of hash function which considers the integer type of information in the program is efficiency and better for data filtering. However, the integer storage space only limits to be four bytes. The EPC code field length exceeds the size, so we use the hash function to generate the 4-byte hash value for data filtering. For example, there is a 96-bit EPC code (shown in Figure 2), and we divided the code into four equal parts. The size of each part is 24-bit, so that each part is not more than 30 bits. Therefore, the size of parts is within the acceptable range of operation. Then the first part (EPC1) and second part (EPC2) produce the first hash value (Hash1), and the third part (EPC3) and fourth part (EPC4) produce the second hash value (Hash2). Finally, the two hash values (Hash1, Hash2) produce a new hashes hash value (Hash3) which is the EPC code for data filtering. Hash coding through the cutting operation, which can effectively shorten the code length to accelerate the data matching the filter efficiency.

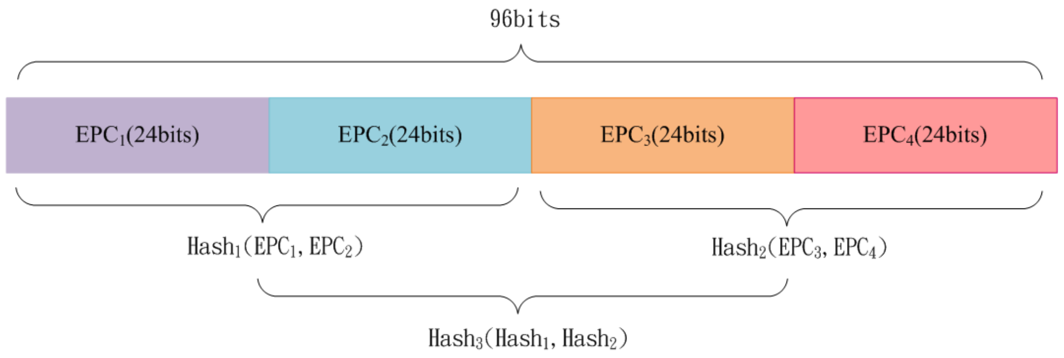


Figure 2. EPC Code of the Hash Function Operation

The tree structure with the hash function filtering method considers two factors. First factor is that each code after filtering has to be converted to URN format. Second factor is that the character of EPC codes can be divided. Therefore, there are a lot of application programs reading the front field of EPC code (shown in Figure 3) repetitively. In this study, we consider the above two factors and the format of EPC code to design the method which is combined with a hash function and the concept of the tree filtering. To speed up the filtration rate, this study divides each encoding type field to store in an integer, but the integer storage space only limits to be four bytes. Some of the fields are longer than the limited size. For example, the EPC code in a certain field length is more than 30 bits, while its hash function operations. In SGTIN-96 specification, when serial and company fields store more than the size of an integer, these fields will be split into two equal parts and use a hash function to split into two integers after the hash function operations to shorten the length of the data. This section is shown in Figure 4.

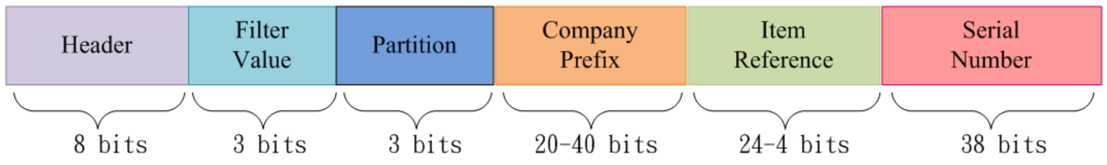


Figure 3. EPC-SGTIN96 (a)

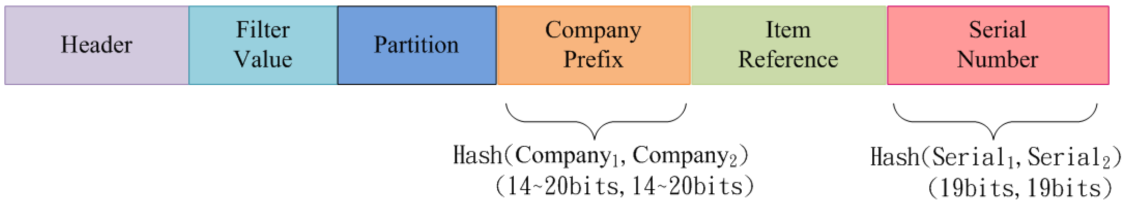


Figure 4. EPC-SGTIN96 (b)

The code data is stored into the tree, and the tree is a basis to find whether the duplicate coding. The EPC code tree segmentation results for the node field units are shown in Figure 5. SGTIN-96 can be divided into six columns. There are the two columns (i.e., the field of company and serial) over integer size. These columns will be computed by the hash function to shorten the field, and the tree stores the hash value calculation.

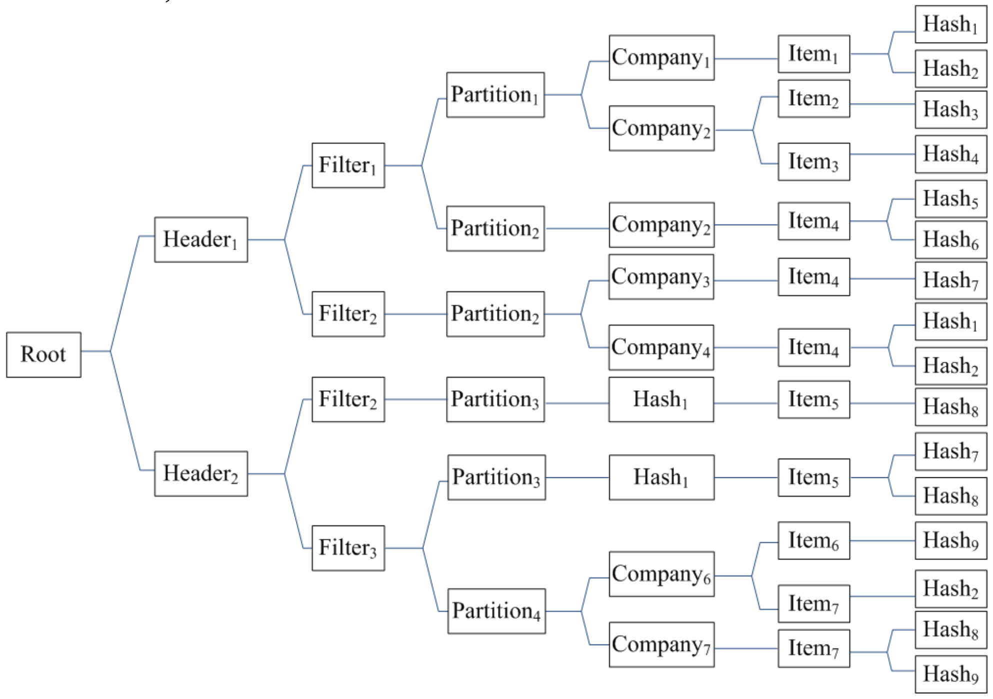


Figure 5. EPC Code Stored by the Tree Field

The tree is built from root node to generate its child nodes until the leaf nodes. For EPC applications, the nodes in upper layer of the tree are read many times, but there are only serial, item and company fields which are different. Therefore, we design the algorithm which searches the node from leaf node to root node. In this algorithm, the number of comparisons is lesser than using the previous algorithms.

4.2. User-defined Filtering

The RFID data are used in different application situations repeatedly. Thus, different application situations require different filtering data. User-Defined Filtering is designed by users, which is mean the users can control the data of duplicate filter, and user also can transform data to information according to the application situation.

Based on the following information and inquiry types, users can customize information filtering and search functions.

4.2.1. Classified Information

Information can be classified according to EPC fields (e.g., header, filter value, company, and item reference).

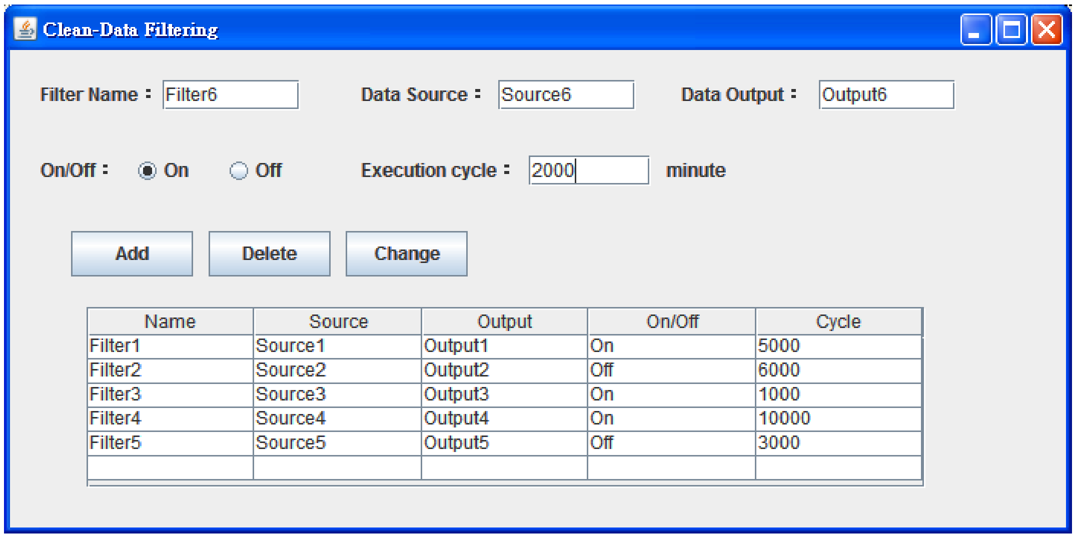
Filtering is designed according to user needs and information not required, or only retains the information needed (e.g., header, filter value, company, and item reference).

4.2.2. Inquiries by the Respective Need

The queries are built according to the EPC field (e.g., header, filter value, company, and item reference).

5. System Implementation

In this study, the concept of RFID intermediary platform is designed within an intermediary platform in RFID, RFID data processing method and data flow design. And we provide the users an interface. The users can set Clean_Data_Filtering and User_Define_Filtering and depend on their application scenarios and conditions to set multiple filter sources for information reusing. The operation screens are as shown in Figure 6 and Figure 7.



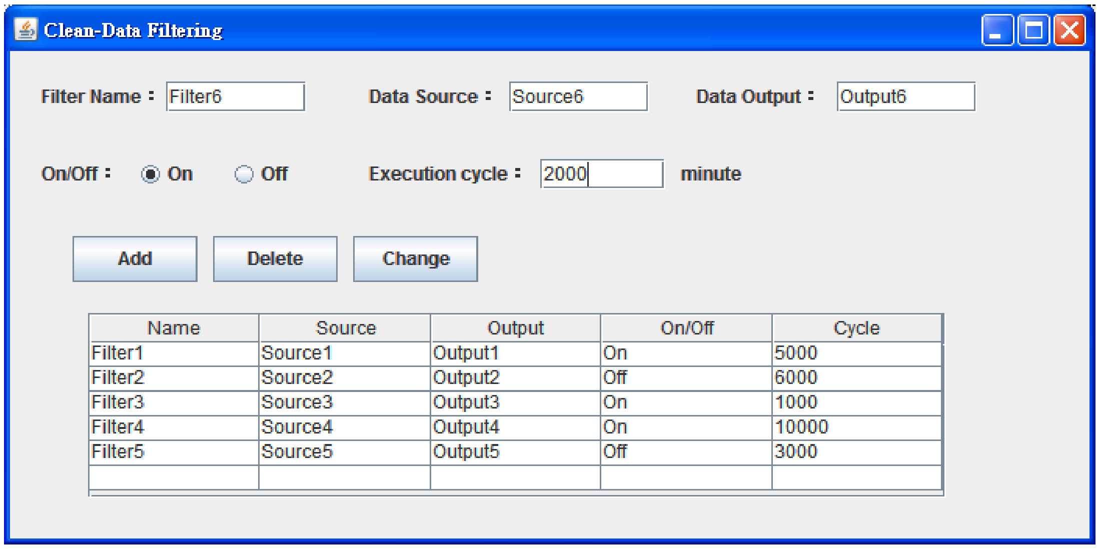
Filter Name : Filter6 Data Source : Source6 Data Output : Output6

On/Off : [✓ On] Off Execution cycle : 2000 minute

Name	Source	Output	On/Off	Cycle
Filter1	Source1	Output1	On	5000
Filter2	Source2	Output2	Off	6000
Filter3	Source3	Output3	On	1000
Filter4	Source4	Output4	On	10000
Filter5	Source5	Output5	Off	3000

Figure 6. Clean_Data Filtering Interface

User_Define Filtering of the output can be divided into the database and XML format. In this paper, we use Microsoft Access to implement the system. Moreover, we provide the XML output format to support the heterogeneous databases in the different platforms.



Filter Name : Filter6 Data Source : Source6 Data Output : Output6

On/Off : [✓ On] Off Execution cycle : 2000 minute

Name	Source	Output	On/Off	Cycle
Filter1	Source1	Output1	On	5000
Filter2	Source2	Output2	Off	6000
Filter3	Source3	Output3	On	1000
Filter4	Source4	Output4	On	10000
Filter5	Source5	Output5	Off	3000

Figure 7. User_Define Filtering Interface

6. Experimental Analysis and Exploration

6.1. Verify of Hash Function

We improve the *minimal perfect hash* (MPH) function for using EPC code [12] shown as formula (2). The MPH algorithm improves the DHP algorithm to avoid the hash collision problems. The parameters X and Y in MPH function H (X, Y) must be unique. However, the EPC codes in this research cannot work in this limitation. Therefore, we improve the MPH function to propose a new hash function which can generate a perfect Hash table and is proved by Reductio ad absurdum.

We prove that the hash value is generated by the hash function $H(X,Y)$ (shown as formula (1)) and is unique by the following verification process.

$$H(X, Y) = 2N(X-1) + Y - X(X-1)/2 \qquad (1)$$

$$H(X, Y) = (X-1)(N-1) + Y - X(X-1)/2 - 2 \qquad (2)$$

Definition: $H(X_1,Y_1) = H(X_2,Y_2)$

i. If this equation is established, then the hash values which are generated by $H(X, Y)$ function are unique.

ii. If this equation does not hold constant, then the hash values which are generated by $H(X, Y)$ are duplicate values.

Rule 1:

i. $0 \leqq X_1, X_2, Y_1, Y_2 \leqq N$

ii. X_1, X_2, Y_1, Y_2 is a positive integer or zero

iii. N is a positive integer.

Rule 2:

i. $X_1 = X_2$ and $Y_1 \neq Y_2$

ii. $X_1 \neq X_2$ and $Y_1 = Y_2$

Hypothesis 1:

$X_1=N-a_1$, $Y_1=N-b_1$, and Substitute $H(X_1,Y_1)$
$H(X_1,Y_1)=2N(N-a_1-1)+ N-b_1-(N-a_1)(N-a_1-1)/2$

Hypothesis 2:

$X_2=N-a_2$, $Y_2=N-b_2$, and Substitute $H(X_2,Y_2)$
$H(X_2,Y_2)=2N(N-a_2-1)+ N-b_2-(N-a_2)(N-a_2-1)/2$

The rules are defined by the following principles:

Principles 1:

i. $a_1, a_2, b_1, b_2,$ *and* N are positive integer

ii. $0 \leqq a_1, a_2, b_1, b_2 \leqq N$

Principles 2:

i. if $a_1 = a_2$ then $b_1 \neq b_2$

ii. if $b_1 = b_2$ then $a_1 \neq a_2$.

The X_1, X_2, Y_1, *and* Y_2 are substituted into $H(X_1,Y_1) = H(X_2,Y_2)$, and verify that the results are following three conditions.

i. $a_1 = a_2$ and $b_1 \neq b_2$

ii. $a_1 \neq a_2$ and $b_1 = b_2$

iii. $a_1 \neq a_2$ and $b_1 \neq b_2$

Simplify $H(X_1, Y_1) = H(X_2,Y_2)$

➔ $2N(N-a_1-1)+ N-b_1-(N-a_1)(N-a_1-1)/2 = 2N(N-a_2-1)+ N-b_2-(N-a_2)(N-a_2-1)/2$

➔ $2N^2-2a_1N-N-b_1-(N^2-2a_1N+a_1^2-N+a_1)/2 = 2N^2-2a_2N-N-b_2-(N^2-2a_2N+a_2^2-N+a_2)/2$

➔ $4N^2-4a_1N-2N-2b_1-N^2+2a_1N-a_1^2+N-a_1 = 4N^2-4a_2N-2N-2b_2-N^2+2a_2N-a_2^2+N-a_2$

➔ $-2a_1N-2b_1-a_1^2-a_1 = -2a_2N-2b_2+-a_2^2-a_2$

➔ $-2a_1N -2b_1-a_1^2-a_1+2a_2N+2b_2+a_2^2+a_2=0$

➔ $+2a_2N -2a_1N +a_2^2-a_1^2+a_2-a_1+2b_2-2b_1 =0$

➔ $(2a_2N-2a_1N)+(a_2^2-a_1^2)+(a_2-a_1)+2(b_2-b_1)=0$

➔ $2N(a_2-a_1)+(a_2-a_1)(a_2+a_1)+(a_2-a_1)+2(b_2-b_1)=0$

➔ $(a_2-a_1)(2N+a_2+a_1+1)+2(b_2-b_1)=0$---$\boxed{1}$

(1) Case 1: if $a_1 = a_2$ *then* $b_1 \neq b_2$

Due to $(a_2-a_1)(2N+a_2+a_1+1)+2(b_2-b_1)=0$, $\boxed{1}$ *simplify* $2(b_2-b_1)=0$

$\because b_1$ *and* b_2 *are positive integer or zero, where* $b_1 \neq b_2$

∴$\boxed{1}$ *No solution.*

(2) *Case 2: if* $b_1=b_2$ *then* $a_1\neq a_2$

Due to $b_2-b_1=0$, $\boxed{1}$ simplify $(a_2-a_1)(2N+a_2+a_1+1)=0$

∵a_1 *and* a_2 *are positive integer or zero,* $a_1\neq a_2$

∴$a_2-a_1\neq 0$ *and* $2N+a_2+a_1+1=0$

∵a_1 *and* a_2 *are positive integer or zero.* N *is a positive integer, so* $2N+a_2+a_1+1>0$

∴$\boxed{1}$ *No solution.*

(3) *Case 3: if* $a_2>a_1$ *then* $b_2>b_1$

Due to a_1, a_2, b_2, and b_1 are positive integer or zero, and N is a positive integer.

∵ $(a_2-a_1)(2N+a_2+a_1+1)>0$ *and* $2(b_2-b_1)>0$

∴$\boxed{1}$ *No solution.*

(4) *Case 4: if* $a_2<a_1$ *then* $b_2<b_1$

Due to a_1, a_2, b_2, and b_1 are positive integer or zero, and N is a positive integer.

∵ $(a_2-a_1)(2N+a_2+a_1+1)<0$ *and* $2(b_2-b_1)<0$

∴$\boxed{1}$ *No solution.*

(5) *Case 5: if* $a_2>a_1$ *then* $b_2<b_1$

Due to $0<a_1, a_2, b_1, b_2\leqq N$

Get $0<(a_2-a_1)\leqq N$ *and* $2N+1<(2N+a_2+a_1+1)<4N+1$

$\rightarrow 2N+1<(a_2-a_1)(2N+a_2+a_1+1)<N(4N+1)$ --$\boxed{2}$

Get $-2N<2(b_2-b_1)\leqq -2$--$\boxed{3}$

∵*The minimum value of* $\boxed{2}$ *is 2n+1 and the maximum value of* $\boxed{3}$ *is -2.*

∴$\boxed{2}>\boxed{3}$. *Therefore, we can see the two functions which are not intersection, where* $(a_2-a_1)(2N+a_2+a_1+1)+2(b_2-b_1)\neq 0$

∴$\boxed{1}$ *No solution.*

(6) *Case 6: if* $a_2<a_1$ *then* $b_2>b_1$

Due to $0<a_1, a_2, b_1, b_2\leqq N$

Get $-N\leqq (a_2-a_1)<0$ *and* $2N+1<(2N+a_2+a_1+1)<4N+1$

$\rightarrow -N(4N+1)<(a_2-a_1)(2N+a_2+a_1+1)<-(2N+1)$ ---$\boxed{4}$

Get $2<2(b_2-b_1)<2N$--$\boxed{5}$

∵*The maximum value of* $\boxed{4}$ *is* $-(2N+1)$ *and the minimum value of* $\boxed{5}$ *is 2.*

$\therefore \boxed{5} > \boxed{4}$. *Therefore, we can see the two functions are not intersection, where* $(a_2-a_1)(2N+a_2+a_1+1)+2(b_2-b_1)\neq 0$

$\therefore \boxed{1}$ *No solution.*

In this paper, we prove that the hash value is generated by the hash function $H(X,Y)$ and is unique by the above verification process. The hash function is $H(X,Y)=2N(X-1)+Y-X(X-1)/2$, X and Y are positive integer or zero, and N is a positive integer. In $0 \leqq (X,Y) \leqq N$ case, the hash values are generated by $H(X,Y)$ are unique.

6.2. Filtering Performance Verification Method

In this paper, we use simulation to verify that the method is efficiency. In experiments, we design two data sets which include: (1) generating 96-bit random numbers and (2) using SGTIN-96 specification.

6.3. Filtration Efficiency Simulations

We consider three cases which include: (1) Duplicate 1/4, (2) Duplicate 1/2, and (3) Duplicate 3/4 shown in table 1. And we then analyze the execution time (milliseconds) which is generated by using four filtering methods (i.e., Cut_Hash, HashCode, Location_filter, and Tradition) with two data sets. Moreover, we consider two situations which include (1) not consider whether analytical code and (2) parse the code and then filtered are described as the following subsections.

Table 1. Number of Data and Duplicate Rate Contrast Table

Number of data	**1000**	**2000**	**3000**	**4000**	**5000**
Duplicate 1/4	750	1500	2250	3000	3750
Duplicate 1/2	500	1000	1500	2000	2500
Duplicate 3/4	250	500	750	1000	1250

6.3.1. Not Consider whether Analytical Code

6.3.1.1. SGTIN Data: Figure 8, Figure 9, and Figure 10 show the HashCode method is the fast in any cases. In case (1), the performances of using Cut_Hash and Location_filter methods are faster with the number of data records which is larger than 2000. In case (2), when the number of data records is larger than 3000, the performances of using Cut_Hash and Location_filter methods are faster than using Tradition method. Moreover, the performance of using Location_filter is faster than using Cut_Hash when the number of data records is larger than 4000. In case (3), the performances of using Cut_Hash and Location_filter methods are slower than using Tradition method.

6.3.1.2. Random Data: For using random data set, Figure 11, Figure 12, and Figure 13 show the HashCode method is the fast in any cases. However, the performances of using Cut_Hash and Location_filter methods are slower in any cases than using Tradition method, and the performance of using Cut_Hash method is faster than using Location_filter method.

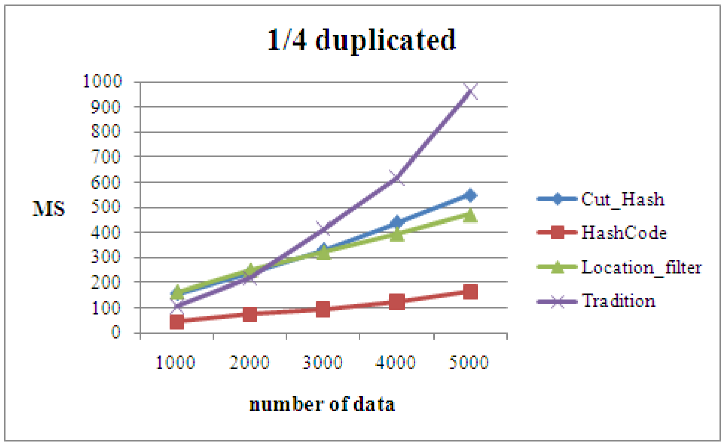


Figure 8. SGTIN Duplicated 1 / 4 Performance Comparison Chart

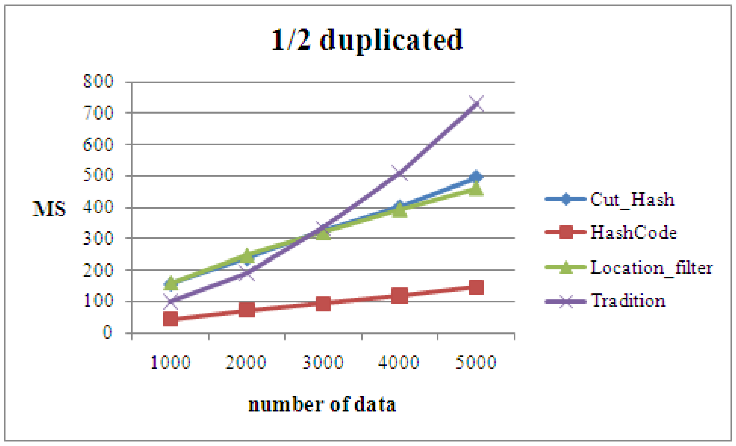


Figure 9. SGTIN Duplicated 1 / 2 Performance Comparison Chart

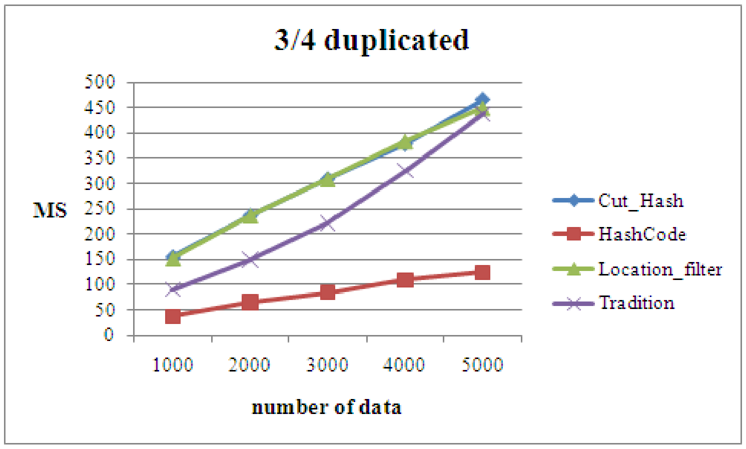


Figure 10. SGTIN Duplicated 3 / 4 Performance Comparison Chart

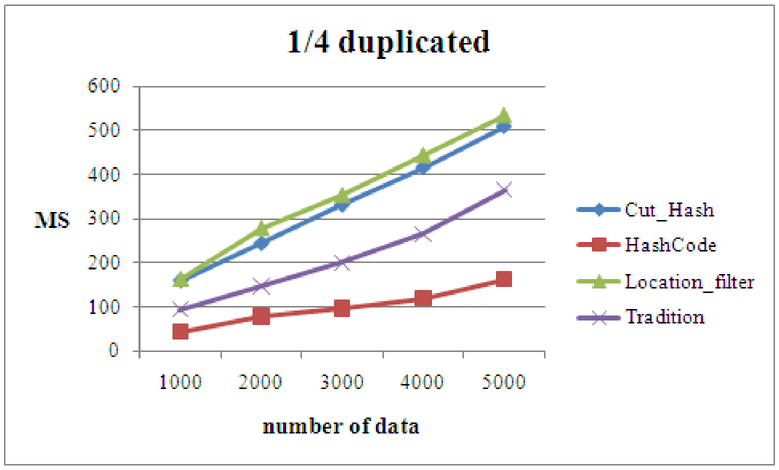


Figure 11. Random Duplicated 1 / 4 Performance Comparison Chart

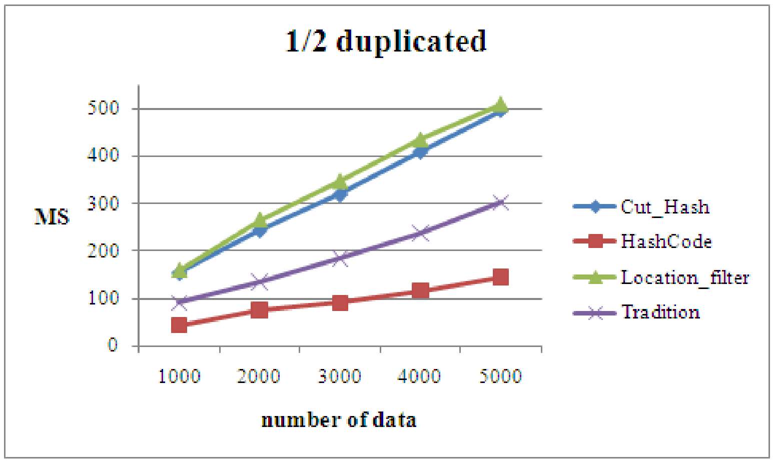


Figure 12. Random Duplicated 1 / 2 Performance Comparison Chart

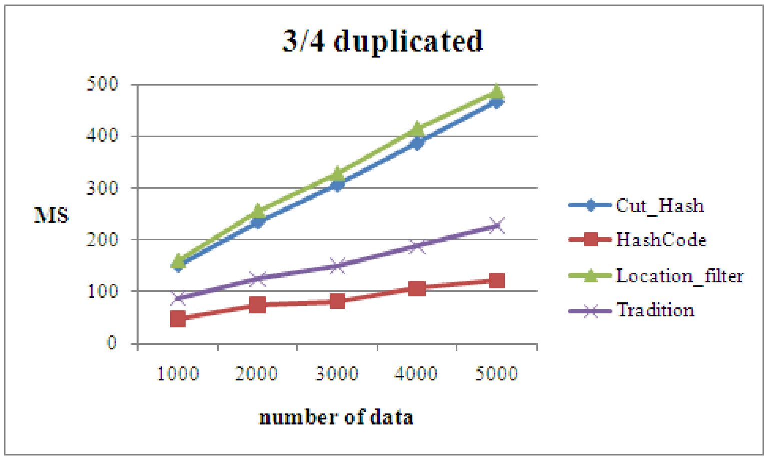


Figure 13. Random Duplicated 3 / 4 Performance Comparison Chart

6.3.2. Parse the Code and then Filtered

6.3.2.1. SGTIN Data: For using SGTIN data set, Figure 14, Figure 15, and Figure 16 show the performances of using Cut_Hash and HashCode methods are faster in any cases. Moreover, the performances of using Cut_Hash and HashCode methods are similarity in each case. Because the encoding actions are built in Location_filter method, the performances of using Location_filter method are only faster than using Tradition method in case (1) and case (2).

6.3.2.2. Random Data: For using random data set, Figure 17, Figure 18, and Figure 19 show the performances of using Cut_Hash and HashCode methods are faster in any cases. Moreover, the performances of using Cut_Hash and HashCode methods are similarity in each case. However, the performance of using Location_filter methods is slower in any cases than using Tradition method.

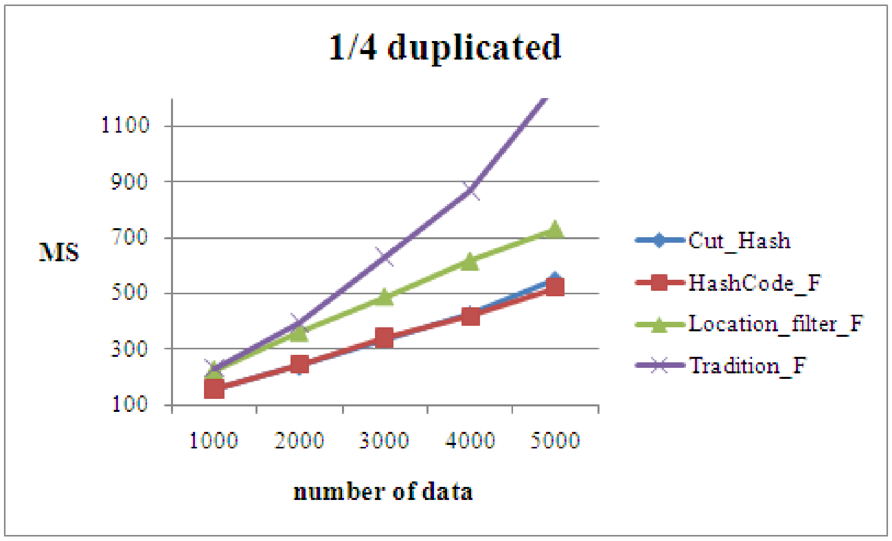


Figure 14. SGTIN Duplicated 1 / 4 Performance Comparison Chart

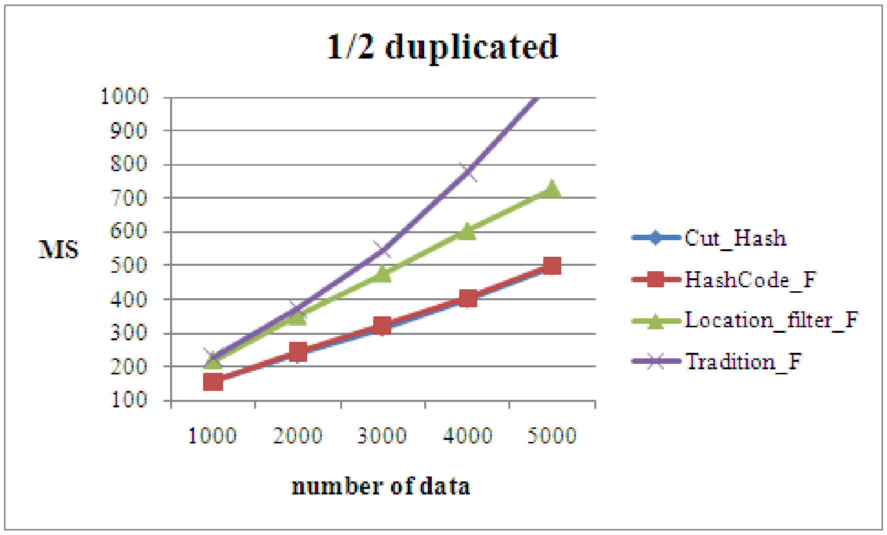


Figure 15. SGTIN Duplicated 1 / 2 Performance Comparison Chart

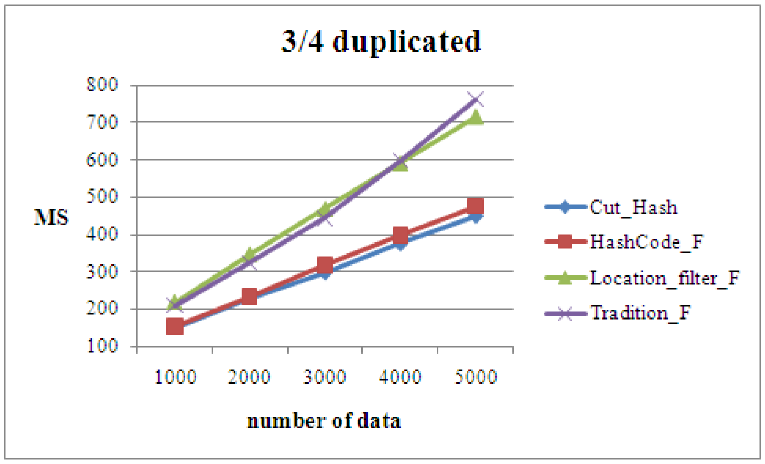


Figure 16. SGTIN Duplicated 3 / 4 Performance Comparison Chart

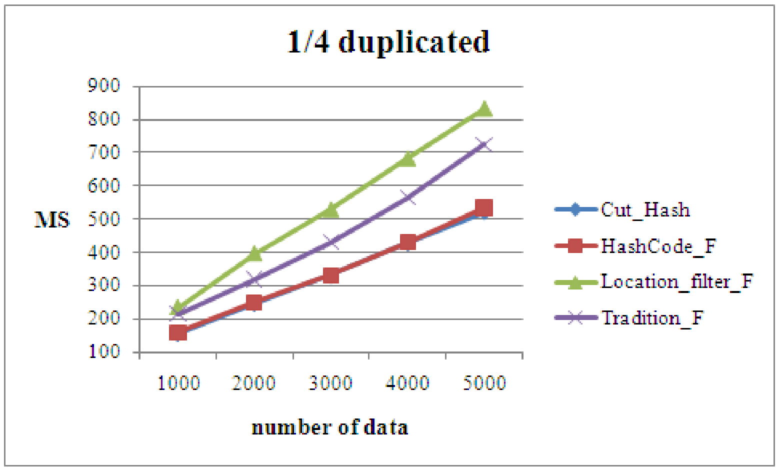


Figure 17. Random Duplicated 1 / 4 Performance Comparison Chart

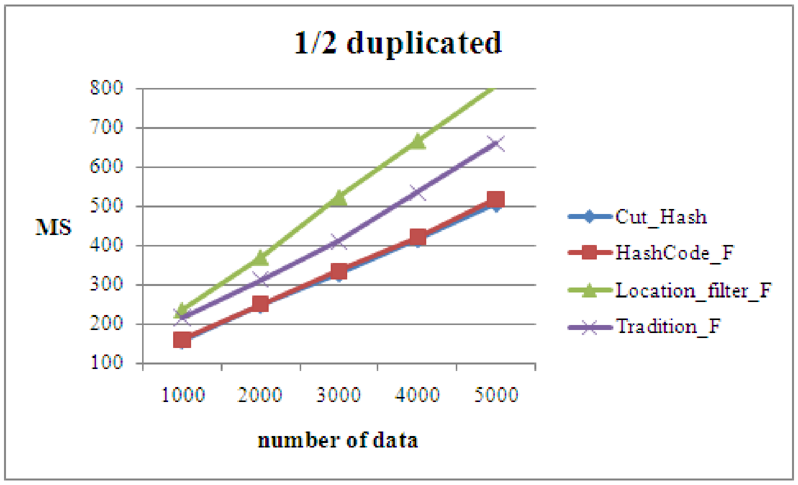


Figure 18. Random Duplicated 1 / 2 Performance Comparison Chart

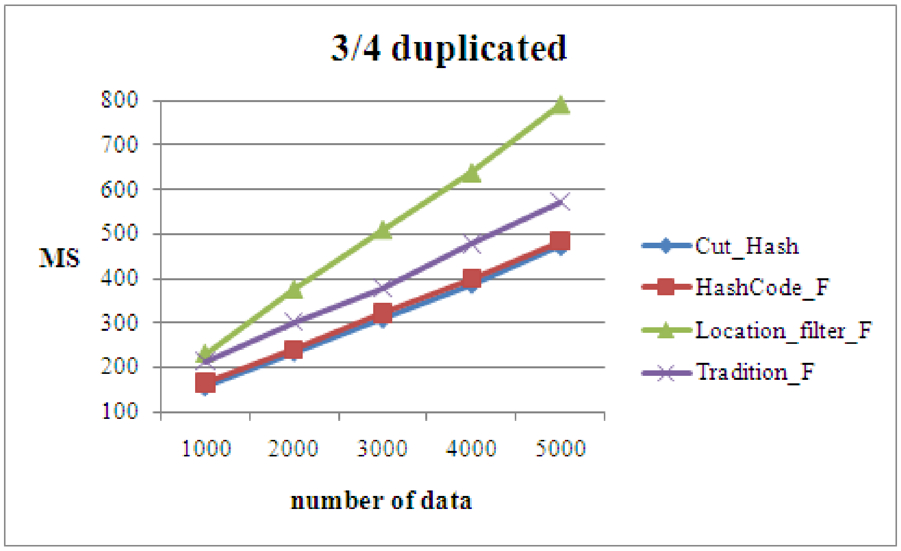


Figure 19. Random Duplicated 3 / 4 Performance Comparison Chart

7. Conclusions

In this study, we improve the MPH function to propose a new hash function which can generate a perfect Hash table and is proved by Reductio ad absurdum. In experiments, we use simulation to verify that the method is efficiency. We consider two situations which include (1) non-considering whether analytical code and (2) parsing the code and then filtered are described as the following subsections. In situation (1), the performance of using HashCode method is the fast in any cases. In situation (2), the performances of using Cut_Hash and HashCode methods are faster than other methods in any cases. Therefore, if the application doesn't consider security issues, using HashCode method is better when. Otherwise, using Cut_Hash method is better.

Acknowledgements

The research is supported by the National Science Council of Taiwan under the grant NSC100-2218-E-020-001 and NSC100-2218-E-020-002.

References

[1] Anagnostopoulos AP, Soldatos JK and Michalakos SG, "REFiLL: A lightweight programmable middleware plafform for cost effective RFID application development", Journal Pervasive and Mobile Computing, vol. 5, no. 1, **(2009)**, pp. 49-63.

[2] Chen CH, Lin BY, Lin CH, Liu YS and Lo CC, "A Green Positioning Algorithm for Campus Guidance System", International Journal of Mobile Communications, vol. 10, no. 2, **(2012)**, pp. 119-131.

[3] Chen CH, Kung HY, Wu CI, Cheng DY, Lo CC, "The Adaptable Food Mileage Services Inference Based on Vehicular Communication", Advanced Science Letters, vol. 5, no. 1, **(2012)**, pp. 101-108.

[4] Cui JF and Chae HS, "Mobile Agent based Load Balancing for RFID Middlewares", In Proceedings of the 33rd Annual Conference on Industrial Electronics Society, **(2007)**.

[5] EPCglobal, "EPC Tag Data Standard", Version 1.5, GS1 EPCglobal, **(2010)**.

[6] Hsiao HC, "A Filter Strategy Using Bit Information in RFID", Master Thesis, Department of Computer Science and Information Engineering, National Dong-Hwa University, **(2008)**.

[7] Kung HY, Chen CH and Ku HH, "Designing Intelligent Disaster Prediction Models and Systems for Debris-Flow Disasters in Taiwan", Expert Systems with Applications, vol. 39, no. 5, **(2012)**, pp. 5838–5856.

[8] Kung HY, Wu CI, Chen CH and Lan YH, "Using RFID Technology and SOA with 4D Escape Route", Proceedings of the 4th IEEE International Conference on Wireless Communications, Network and Mobile Computing (WiCOM 2008): Information Systems and Management, **(2008)**.

[9] Liu Z, Liu F and Lin K, "Agent-based Device Management in RFID Middleware", In Proceedings of the 4th International Conference on Wireless Communications, Networking and Mobile Computing , **(2008)**.

[10] Lo CC, Chen CH, Cheng DY and Kung HY, "Ubiquitous Healthcare Service System with Context-awareness Capability: Design and Implementation", Expert Systems with Applications, vol. 38, no. 4, **(2011)**, pp. 4416-4436.

[11] Mo JPT, Sheng QZ, Li X and Zeadally S, "RFID Infrastructure Design; A Case Study of Two Australian RFID Projects", IEEE Internet Computing, vol. 13, no. 1, **(2009)**, pp. 14-24.

[12] Tsai WF, "A Minimal Perfect Hashing and Pruning Approach for Mining Association Rules", Master Thesis, Department of Information Management, National Chi Nan University, **(2002)**.

Authors

Sih-Ying Li received her BS degree from Department of Management Information Systems of National Pingtung University of Science and Technology, Taiwan, in 2010. Now, she is working her MS. Degree at the same University. Her research interests include wireless and mobile communications, and multimedia applications.

Hsu-Yang Kung received his BS degree from Tatung University, MS degree from National Tsing-Hwa University, PhD degree from National Cheng-Kung University, Taiwan, all in computer science and information engineering. He is currently a professor and Dean of College of Management, National Pingtung University of Science and Technology, Taiwan. Prof. Kung published more than 200 academic papers. Prof. Kung received the Excellent Research Group Awards 4 times from National Science Council and the Excellent Research Award NPUST at 2010. His research interests include cloud computing, wireless and mobile communications, and embedded multimedia applications.

Chi-Hua Chen received a B.S. degree from National Pingtung University of Science and Technology, Taiwan, in 2007, and an M.B.A. degree from National Chiao Tung University, Taiwan, in 2009, all in information Management. He is currently working toward the Ph.D. degree with the Institute of Information Management, National Chiao Tung University, Hsinchu, Taiwan. He has published over 70 academic papers. His recent research interests are in cloud computing, cellular network, data mining, intelligent transportation system, network security, healthcare system, and E-learning System.

Wen-Hsi Lydia Hsu received her Master degree with first class honours and PhD degree from Lincoln University, New Zealand. She is currently an Assistant Professor at the Department of Business Administration, National Pingtung University of Science and Technology, Taiwan. Her research interests focus on corporate governance, information technology governance, cloud computing governance, Enterprise Resource Planning and financial accounting.