Collision evasion method using auxiliary hash

Jeonghyeok Kim, Jaemin Hwang, Joohyeong Song, Jongsik Lee and Sanggil Kang

Abstract— **In this paper, we develop a conflict-free index generator to increase performance in a big data environment. There are several problems with the conventional hash functions, such as Minimal Perfect Hash Function (MPHF) on dynamic systems like big data. The collision-free problem occurs with an increase in the amount of data and the overhead in securing additional space to solve this problem. To solve this problem, we propose a collision evasion method using an auxiliary hash. In this paper, we divide the data into two categories by constructing a double hash to solve the problem.**

Keywords—**big data, conflict free, hash function, index generation function**

I. Introduction

Databases (DBs) for information systems have developed rapidly since the 1990s. In particular, the DB plays a core role in a company's information systems, such as enterprise resource planning (ERP) [1], customer relationship management (CRM) [2], and in data warehouses [3]. Also, the DB is now used for big data systems. The most important element is the index generation function for DB performance improvement. The index generation function is a multi-valued logic function that checks whether the given input vector is registered or not, returning its index value if the vector is registered. Index generation latency due to overhead is critical. Minimal Perfect Hash Function (MPHF) [4] is one of the index generation functions, which is widely used for index generation. However, MPHF has two problems. First, MPHF needs to restrict the input range in order to prevent data conflict, but the input range is not defined or limited in big data systems. Second, additional overhead is required to control data conflict. This is not efficient for a big data system.

To solve this problem, we propose a collision evasion method using an auxiliary hash. The remainder of this paper is organized as follows. Section 2 explains related works. In Section 3, we describe our proposed collision evasion method using an auxiliary hash. Section 4 demonstrates the collision evasion method using an auxiliary hash. We conclude our work in Section 5.

II. Related Work

Studies of index generation functions have been widely carried out. Index generation functions have been utilized for various purposes, such as linear transformation, hash, and so on. Index generation functions are just logic functions with n inputs and ⌈log2 (k + 1) ⌉ outputs, where k is a registered vector. Recall that the special value of '0' should be included among the output values. Many researchers might use ordinal logic synthesis methods. However, the normal logic synthesis method is not good for random functions. Additionally, from an application point of view, registered vectors may change, and fixed gate circuits do not handle that situation. Reconfigurable circuits such as field-programmable gate arrays [5] could handle them, although they have overhead resynthesizing the circuits. A naive way to implement index generation functions is to use memory with an address line. Of course, this is also not a good way. In many cases, the number of registered vectors (= k) is far less than 2n, so there would be a lot of '0' entries in memory [6].

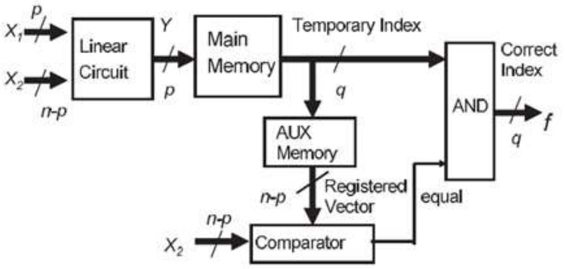

Figure 1 Index generation unit

Figure 1 shows an index generation unit (IGU) [7]. The linear circuit has n inputs and p outputs, where p < n. It produces the following functions:

$$y_1 = c_{1,1}x_1 \oplus c_{1,2}x_2 \oplus c_{1,3}x_3 \oplus \cdot \quad \cdot \quad \cdot \oplus c_{1,n}x_n$$

$$y_2 = c_{2,1}x_1 \oplus c_{2,2}x_2 \oplus c_{2,3}x_3 \oplus \cdot \quad \cdot \quad \cdot \oplus c_{2,n}x_n$$

$$y_3 = c_{3,1}x_1 \oplus c_{3,2}x_2 \oplus c_{3,3}x_3 \oplus \cdot \quad \cdot \quad \cdot \oplus c_{3,n}x_n$$

$$y_p = c_{p,1}x_1 \oplus c_{p,2}x_2 \oplus c_{p,3}x_3 \oplus \cdot \quad \cdot \quad \cdot \oplus c_{p,n}x_n,$$

where $c_{i,j} \in \{0, 1\}$, and $c_{i,i} = 1$. It is used to reduce the size of main memory. Let X1 = $(x_1, x_2, . . . , x_p)$ and X2 = $(x_{p+1}, x_{p+2}, . . . , x_n)$.

Main memory has p inputs and log2 (k+1) outputs. Main memory produces correct indices only for registered vectors. However, it may produce incorrect indices for non-registered

Jeonghyuk Kim, Jaemin Hwang , Joohyeong Song, Jongsik Lee and Sanggil Kang
Inha University
Republic of Korea

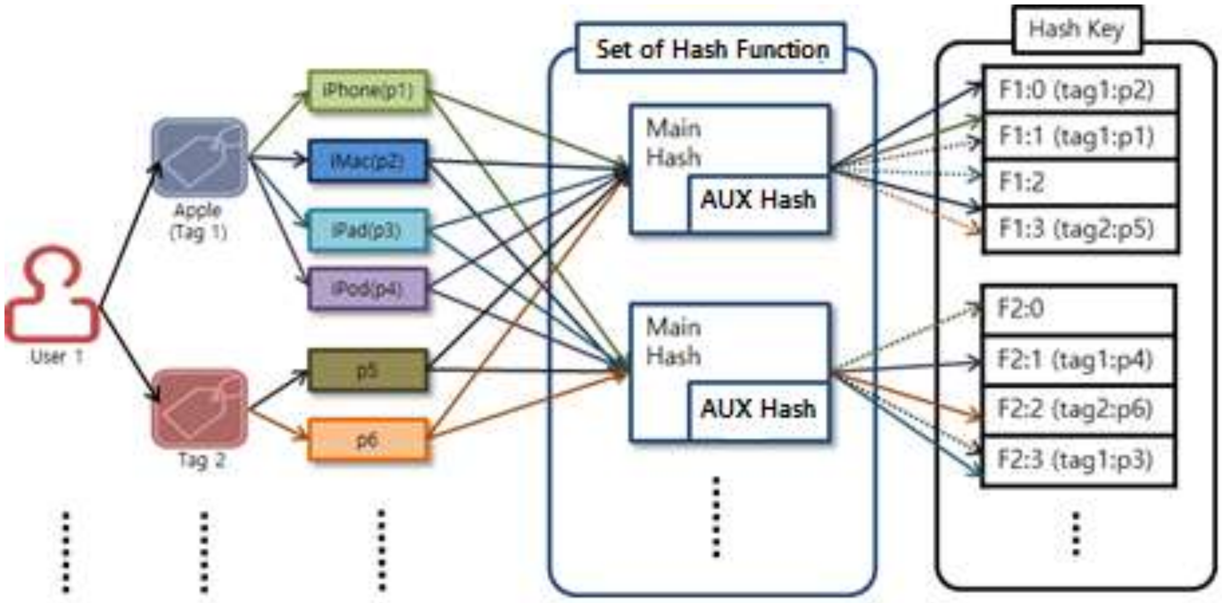

Figure 2. Mainstream structure of our paper

vectors, because the number of input variables is reduced by using "don't care" conditions. In an index generation function, if the input vector is non-registered, then it should produce 0 output. To check whether main memory produces the correct index or not, we use AUX memory. AUX memory has log2 (k +1) inputs and n − p outputs: it stores the X2 part of registered vectors for each index. The comparator checks whether the X2 part of the input is the same as the X2 part of the registered vector. If they are the same, main memory produces a correct index. Otherwise, main memory produces an incorrect index, and the input vector is non-registered. In this case, the output AND gates produce 0, showing that the input vector is non-registered.

BLAKE is one candidate algorithm of Secure Hash Algorithm 3 (SHA-3), meeting all standards set by the National Institute of Standards and Technology, providing security guarantees in theory and in practice. BLAKE is built on the LAKE hash function family, with HAIFA as the iteration mode and a local wide-pipe as its internal structure, with the ChaCha [8] stream cipher as its core function. BLAKE resists generic second-preimage attacks, length-extension and side channel attacks [9].

However, these studies have two problems. First, the normal logic synthesis method is not good for random functions. Second, a hash function needs additional overhead to control the data conflict. To solve these problems, we designed a collision evasion method using an auxiliary hash. In the following section, we propose the collision evasion method using an auxiliary hash.

III. Collision evasion method

Figure 2 is the structure of our paper to evade an item collision via hash. The structure consists of three parts: methods of determining the optimum hash number, configuring matches between item attributes and hashes, and a method for determining the optimum index number.

Figure 2, as shown in the rightmost block constitutes a majority of one or another independent hash functions that can accommodate the entire index. For example, item p1(iPhone) gets hash F1 and hash F2 in the Collision Free Indexing (CFI) process, then reaches hash keys (F1: 1) and (F2: 0). Finally, item p1 gets a unique index (F1: 1), as shown by the solid line in the figure. Thus, in order to provide a hash key only for every item p, it is necessary to specify the hash number and the size of the index.

First, we discuss methods of determining the optimum hash number.

It may be necessary to predict the hash through maximum matching of bipartite graphs [10]. First, the input data set makes a bipartite graph G hash function, where the resulting set consists of P and F. Second, find the maximum matching for G using the Hopcroft-Karp algorithm [11]. Third, if G is the same size as the maximum match for P, then dividing the F function matches. Last, if it is not the same, reconstitute G with additional hash functions.

Next, configure matching between item attributes and hashes.

First, f (,) is defined as a hash function where it is the set of main item t and the set of a sub item p.

1. {X1} = (x_1, x_2, ...,x_q) and {X2} = ($x_{(q+1)}$, $x_{(q+2)}$, ...,x_n), where q is ⌈$\log_2$ ((n+1)/3)⌉, and n is the number of all items.

2. While $|X_1| \leq q$, insert the minimum result by $|the\ number\ of\ (x_i = 0) - the\ number\ of\ (x_i = 1)|$ into $\{X_1\}$ from $\{X_2\}$.

3. When randomly compared to replacing any x_i, x_j values in the $\{X_1\}$, $\{X_2\}$ group, if the number of collisions is reduced, then exchange them.

4. Until $|X_1| > q$, repeat steps 2-3.

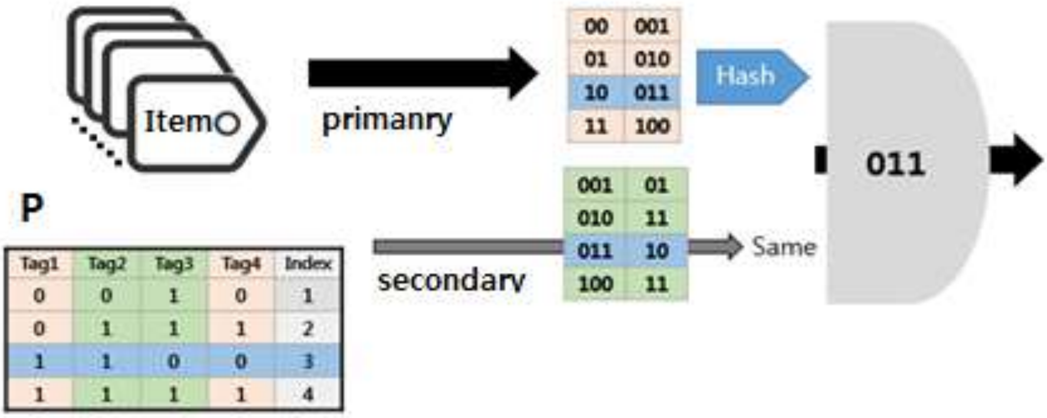

Figure 3 Method for determining the optimum index number

The next step is a method for determining the optimum index number.

If the number of the indices assigned to the hash function is reduced, afterwards, the probability of collision is low, but the index is inefficient because of the presence of unused indices. On the other hand, when the number of indices is too low, that increases the chance of conflicts. Therefore, the conflict should be configured to obtain the minimum number of indices.

Figure 3 shows when the minimum number of item attributes and hash functions is determined, and based on this, indexing can be performed. First, for the input value P, separate primary and secondary characteristic properties. Second, endow main properties with the index via a hash function. Third, compare the sub-values for the index value; if they match, output the index value. Last, if it does not match, the new item attribute is not in the existing index value, so create an additional output after allocation.

IV.Experiment

As shown in Table 1, the equipment used for the CFI experiments were an Intel Core i7-4790 processor with 12GB DDR3 RAM and the Windows 8.1 operating system. Whole key sets were randomly created, with a set of 1 million from a word-collecting bot on the Internet and a set of 1 million collected from a dictionary and web documents; the mean length is 11 bytes, and the maximum length is 50 bytes. The result is the average value measured after the hash function was successfully repeated five times.

Table 1 Experiment Environment

CPU	Intel Core i7-4790
RAM	12G DDR
OS	Windows 8.1

Provided that the input key set is 500,000, the MPHF technique handles 8156.6 key sets per second. With input of two million, MPHF handles 8100 key sets per second. Although the number of input key sets was not significant, with higher numbers, the equipment tended to slow down a little bit.

However, the CFI technique presented in this paper handles 8944.5 key sets per second. Moreover, with input of two million, CFI handles 11396.0 key sets per second. Unlike MPHF techniques, we can see that the CFI process speeds up gradually as the number of input key sets increases.

The above results indicate good performance for big data processing. Compared with the conventional techniques (for example, MPHF), the CFI technique is normally adapted to processing growing amounts of data.

Table 2 MPHF time latency measurement

Key set (ea.)	50	100	150	200
Seek time (sec)	57.9	115.6	173.4	232.5
Time latency result (sec)	61.3	122.4	183.0	246.9

Table 3 CFI time latency measurement

Key set (ea.)	50	100	150	200
Seek time (sec)	52.5	92.6	125.9	161.1
Time latency result (sec)	55.9	99.2	134.6	175.5

Tables 2 and 3 compare the MPHF technique's [12] and the CFI technique's seek time and full-time measurement for time latency for several input levels. The results show similar search times for 500,000, but with increasing numbers of key sets, CFI's time latency is better than MPHF. So, these results may apply to big data environments.

V. Conclusion

DBs and big data are essential to modern information and communications technologies. Accordingly, we describe an indexing scheme to distinguish data items accurately. And we compensate for the problem with conventional MPHF by proposing an auxiliary hash method for conflict-free index generation. This method provides more accurate results for classification of indexed big data, and is expected to reduce the overall cost because it is conflict free.

In experiments, we proposed a structure for constituting a number of hash functions for non-collision hash. Then, we verified that the higher the amount of data, the faster the processing speed. However, we could not check successive addition of data sets for processing big data. Later, we will improve the method in case that problem occurs.

The next study must resolve dynamic configuration for item sizes in a big data environment. At this time, we will have to conduct further studies for improvement in reducing overhead that may occur.

Acknowledgment

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(NRF-2014R1A1A2056374)

References

[1] Jun Han, Rongbi Liu, Brandon Swanner, Shicheng Yang, "ENTERPRISE RESOURCE PLANNING ", umsl.edu, 2010

[2] PC Verhoef "Understanding the effect of customer relationship management efforts on customer retention and customer share development", Journal of marketing, journals.ama.org, 2003

[3] LP English, "Improving data warehouse and business information quality" , cds.cern.ch, 1999

[4] ZJ Czech, G Havas, BS Majewski, "An optimal algorithm for generating minimal perfect hash functions", Information Processing Letters, 1992

[5] T. Sasao, "A Design method of address generators using hash memories," *IWLS-2006*, Vail, Colorado, U.S.A, June 7-9, 2006, pp.102-109.

[6] Ashok Sudarsanam, Sharad Malik. 1995, "Memory bank and register allocation in software synthesis for ASIPs", IEEE/ACM international conference on Computer-aided design (ICCAD '95). IEEE Computer Society, Washington, DC, USA, 388-392.

[7] T. Sasao., "On the number of variables to represent sparse logic functions," *ICCAD-2008*, San Jose, California, USA, Nov.10-13, 2008,. pp 45-51

[8] Salsa20,[EB/OL] http://en.wikipedia.org/wiki/Salsa20.2010 December.

[9] AUMASSON Jean-Philippe, HENZEN Luca, MEIER Willi, et.al, SHA-3proposal BLAKE Round 2 Candidates, 2010 September.

[10] WL Xu, Z Tang, Z Chen, RF Li "Research of new index generation algorithm based on hash function", Software Engineering and Service, 2013

[11] T Sasao. "Index Generation Functions: Recent Developments", ISMVL, 2011 lsi-cad.com

[12] Hagyu Lee, "A Selecting-Ordering-Mapping-Searching Approach for Minimal Perfect Hash Functions", Journal of korean institute of information scientists and engineers, 2000.1, pp 41-49

About Author (s):

Jeonghyeok Kim

received the bachelor's degrees in Computer Science from Inha University, South Korea in 2013, respectively. He is currently a graduate student in the Department of Computer Science and Information Engineering at INHA University, Korea. His research interests include Data mining, Inference Systems, etc.

Jaemin Hwang

received the bachelor's degrees in Computer Science from Inha University, South Korea in 2013, respectively. He is currently a graduate student in the Department of Computer Science and Information Engineering at INHA University, Korea. His research interests include System Architecture, Network, Multimedia Systems, Database, etc.

Joohyung Song

received the bachelor's degrees in Computer Science from Inha University, South Korea in 2015, respectively. He is currently a graduate student in the Department of Computer Science and Information Engineering at INHA University, Korea. His research interests include Data mining, Database, Inference Systems, etc.

JongSik Lee

received the B.S. and M.S. degrees at Department of Electronics Engineering from Inha University in 1993 and 1995. And, he received the Ph.D. degree at Department of Electrical and Computer Engineering from University of Arizona in 2001. His research interests include Software modeling and simulation, Cloud computing, etc.

Sanggil Kang

received the M.S. and Ph.D. degrees in Electrical Engineering from Columbia University and Syracuse University, USA in 1995 and 2002, respectively. He is currently an associate Professor in the Department of Computer Science and Information Engineering at INHA University, Korea. His research interests include Semantic Web, Artificial Intelligence, Multimedia Systems, Inference Systems, etc.