國立交通大學

網路工程研究所

碩 士 論 文

在布隆過濾器下改善範圍搜尋方法

The Bloom Filter Design for Numerical Range Query

研 究 生：范坤揚

指導教授：張明峰　教授

中 華 民 國　九 十 七　年 七 月

在布隆過濾器下改善範圍搜尋方法

The Bloom Filter Design for Numerical Range Query

研 究 生：范坤揚　　Student：Kun-Yang Fan

指導教授：張明峰　　Advisor：Ming-Feng Chang

國 立 交 通 大 學

網 路 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Network Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

July 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年七月

在布隆過濾器下改善範圍搜尋方法

學生：范坤揚　　指導教授：張明峰 博士

國立交通大學網路工程研究所

摘要

布隆過濾器是一個簡單具有空間效率並且隨機的資料結構，被用作簡明的資料集合的表示方法。它的隨機特性具有相當大潛力被應用在分散式網路系統，並且支援系統內的會員資格詢問加上較低的錯誤判正率。布隆過濾器的錯誤判正率是一種事件發生機率，代表一個不屬於資料集合的元素被布隆過濾器判定為屬於資料集合內。目前已經有很多的研究在於如何降低布隆過濾器的錯誤判正率，但是相當少的文獻探討在針對數值範圍搜尋應用下改善布隆過濾器。然而，很少的文獻在探討針對數值範圍詢問的布隆過濾器設計。由於布隆過濾器只能表示有限制的原素個數，當一個大的數值範圍插入布隆過濾器時，錯誤判正率會急劇的升高。在本篇論文中，我們針對數值範圍提出有效率的布隆過濾器的設計。首先，區間方法利用將數值範圍分群到區間的方式來降低插入的元素個數，因此數值在同一區間會被視為同一元素。另一方面，重疊方法利用連續數值的插入位元的重複來降低布隆過濾器的插入位元數。另外，區間和重複的方法結合之前所提到的方法。分析模型被用來找出各個方法的錯誤判正率。電腦模擬被用來驗證我們分析模型的正確性。更進一步，布隆過濾器表示連續範圍之最佳的參數設定可以被得到，因此錯誤判正率會被減到最小。一個啟發式演算法已經被提出用來找在多重屬性下的近似最佳參數設定。區間和重疊的方法針對數值範圍延伸布隆過濾器的設計當傳統的布隆過濾器不能在被使用。

The Bloom Filter Design for Numerical Range Query

Student: Kun-Yang Fan Advisor: Dr. Ming-Feng Chang

Institute of Network Engineering

National Chiao Tung University

ABSTRACT

A Bloom filter is a simple space-efficient randomized data structure for concisely representing a data set. The property of its randomization has great potential for distributed network systems, and it supports the membership query with a small false positive rate, which is the probability that an element was not in the data set but Bloom filter reported it is. There have been many studies on how to improve the correctness of Bloom filter by reducing the false positive rate. However, little research has been done on Bloom filter design for numerical range query. Since a Bloom filter can only represent a limited number of elements, when a large range of numerical attributes are inserted into a Bloom filter, the false positive rate increases dramatically. In this thesis we present efficient Bloom filter design for numerical ranges. First, Division scheme reduces the number of elements inserted by grouping the numerical range into divisions, i.e., numbers in the same division are treated as the same element. On the other hand, Overlapping scheme reduced the number of bits inserted in the Bloom filter by overlapping the inserted bits of consecutive numbers. In addition, Division and Overlapping scheme combines the techniques of the aforementioned two schemes. Analytic model was used to derive the false positive rates of the schemes. Computer simulations were used to verify the correctness of the analytic model.

Moreover, the optimal configuration of Bloom filter representing a numeric range of single attribute can be obtained, i.e., the false positive rate is minimized. A heuristic algorithm has been developed to obtain near optimal configurations for multiple attributes. The Division and Overlapping scheme extends the Bloom filter design for numerical range query, where traditional Bloom filter cannot be used.

誌謝

首先我要感謝張明峰教授這兩年來的費心指導，方能順利完成此篇論文。於求學過程中，在老師的啟發與教誨下，引導學生走向正確的方向，培養獨立思考的能力，著實使我獲益良多。

還有要感謝網際網路通訊實驗室的同學們，冠璋、忠育、君飛及學弟們的支持與鼓勵，給予我碩士生涯中奮鬥不懈的動力和增添不少的色彩，由衷的感謝你們！

最後將此論文獻給我最親愛的家人。感謝您們在我求學期間全心全意的支持，讓我得以專心的完成研究。

Table of Contents

List of Figures

List of Tables

List of Algorithms

List of Functions

Chapter 1 Introduction

1.1 Overview

In recent years overlay and peer-to-peer network applications, such as file sharing, Internet telephony and group communication systems, have been replacing the traditional client-server model. The peer-to-peer network applications use the distributed hash tables to locate a node or object in peer-to-peer network [1], [2]. And each node in peer-to-peer network only preserves a part list of objects locations in a peer-to-peer system instead of every object location in other nodes. The replication of global index is well distributed over peer-to-peer network; therefore, keeping the distributed hash table at each node is important in the moderate-sized peer-to-peer network construction for large-scale scalability.

Bloom filter has been used to profile the description of a node in a P2P systems or a set of data, including numerical and non-numerical items. The PlanetP is a peer-to-peer system that using Bloom filter to summarize the set of data items in peer's local index [3]. As a result, the cost of replication can be reduced and the distributed hash table of peer's local cache would be minimized by compressing the bloom filter. Reynolds and Vahdat demonstrate another application where Bloom filter was used to find the set intersection for keyword searches [4].

Although Bloom filter is a space-efficient way to represent a set of data, it has difficulty in representing a large range of numerical data. Because the large number of inserted items will result in the false positive rate increasing, we need to find more efficient way to insert data into Bloom filter. For example, a numerical rage of a data set contains a class c IP address 140.113.214.X, which includes 255 sub IP addresses. And if we used Bloom filter to represent this data set for previous application [3], the

large number of inserted elements of Bloom filter would make performance degradation of Bloom filter. In this thesis we discuss this kind of numerical range problems and propose our schemes to improve the former methods for numerical ranges.

1.2 Related Work

Several researches have addressed the issues how to improve using space or comparison time of Bloom filter and still maintain a low false positive probability. Bloom filter is a bit array to represent a set of data elements by mapping the set of data into the randomized bit array indices. In other words, the different indices of bit array are set to 1 or 0 to represent a set of data. The false positive occurs when the Bloom filter reports the element x is in the set although it is actually not in the set. In addition, inserting element into Bloom filter changes the probability of false positive. The background on Bloom filter theory is presented in chapter 2.

Fan, Cao, Almerida, and Broder [5] proposed an extending Bloom filter, using counter array to replace the bit array of Bloom filter for inserting and deleting; therefore, it can be more scalable to summary the web server cache. When an element is inserted into the cache, the counter increased from 0 to 1; when an element is deleted from the cache, the counter decreased from 1 to 0. This method avoids the problem that the Bloom filter loses the correctness after inserting or removing element elements because bit counter can dynamically increase or decrease rather than a single bit. Mitzenmacher [6] suggested a Compressed Bloom filter to improve the performance in term of bandwidth saving when the Bloom filters are used to the transmission messages. The method of compressed Bloom filter is to compress the bit array size of Bloom filter and use less number of hash function in Bloom filter. The author emphasized the point that the number of hashing function minimized the false

positive probability in uncompressed Bloom filter case but maximized the probability in the case of Compressed Bloom filter. Kirsch and Mitzenmacher [7] also proposed distance-sensitive Bloom filter, using a set of locality-sensitive hash functions to answer queries of the forms, "Is *x* close to an element of *S*?" It has potentially benefits of the speed of membership query comparisons and requires less space than the original data.

Cohen and Matias [8] proposed spectral Bloom filter and addressed the issue of element deletion over multi-sets of Bloom filter. Spectral Bloom filter is an extension of original Bloom filter to estimate the multiplicities of individual elements with small error probability. Kumar, Xu, Li, and Wang [9] showed another compact structure space-code Bloom filter, which is an approximate representation of a multi-set. Space-code allows for the query about how many occurrences of an element being there in a multi-set. Both Bloom filters are approximate representations of a multi-set, which allows for querying multiplicities of an element. Spectral Bloom filter, space-code Bloom filter and their variations are suitable for representing static sets whose size can be estimated before design and development.

Instead of representing static sets, dynamic Bloom filter [10] and scalable Bloom filter [11] are proposed to dynamic sets when the actual size of a data set increases. Dynamic Bloom filter is a bit matrix with *s* lows and *m* columns. In other words, dynamic Bloom filter consists of *s* standard Bloom filters with length *m*, and it starts with $s = 1$ when no inserting element. When inserting new elements, dynamic Bloom filter may increase the number of rows *s* if it could not find an active bloom filter, and an active Bloom filter of dynamic means that the number of inserting elements does not exceed the threshold of the standard Bloom filter with size *m* for maintaining false positive rate at constant value below. Therefore, the inserting element did not be inserted until finding an active Bloom filter in dynamic Bloom filter or adding a new

standard Bloom filter for an active Bloom filter. Scalable Bloom filter improves the performance degradation of dynamic Bloom filter when the number of standard Bloom filter increases. The main difference between dynamic Bloom filter and scalable Bloom filter is the method of adding a new standard Bloom filter. Scalable Bloom filter is a bit matrix as same as dynamic Bloom filter, but it inserts an active Bloom filter with double size of previous active Bloom filter rather than dynamic Bloom filter. Scalable Bloom filter provides the lower query time and more scalable inserting method than dynamic Bloom filter.

1.3 Objective

Although many studies have been done on the data structure improvement of Bloom filter, little information is available on inserting method over Bloom filter. Previous works have proposed many variations of standard Bloom filter, but it still remains the issue how to efficiently insert element into Bloom filter. The purpose of this thesis was to investigate the effect of inserting many numerical elements, which increases false positive probability, and we will propose our schemes Division, Overlapping and the combination of both Division-Overlapping to improve the method of numerical elements insertion.

In the thesis, we address the issue of numerical range insertion using Bloom filer, and show how a numerical range, which contains many elements, can be represented and stored in a Bloom filter with less space. The representation scheme of our work may increasing the efficiency of Bloom filter in query time and space when numerical elements having a large percentage of a data set. Our contribution of this thesis is to propose an efficient scheme for the mapping from numerical rages to Bloom filer, and we will give our suggestion for the parameters setting of our methods in this thesis.

1.4 Summary

We organized the remaining thesis in the following. Chapter 2 presents the background of Bloom filter theory and the definition of range query. In Chapter 3, we describe our methods in representing a numeric range and the analytic models. In Chapter 4, we evaluate the effectiveness of our methods and discuss the simulation results. Finally we give our conclusion in Chapter 5.

Chapter 2 Background

2.1 Standard Bloom Filter

Bloom filter is a space-efficient randomized data structure to represent a data set. Fig. 2-1 depicts standard bloom filter [12], which is a bit array that represents a set *S* of *n* elements $\{s_1, s_2 \ldots s_n\}$, and it uses a set of $k = 4$ hash functions to project the elements of a data set *S* onto the bit array with a set of $4*n$ random indices. In this figure, the length of this bit array is *m*, and the number of hash functions *k* is 4. The four hash functions of Bloom filter would be independent for randomization property; therefore the hashed indices of element would not be the same. Because of the randomization property of hash function, the four hashed indices of each element in *S* may have different value from each other. After hashing *n* elements in the data set *S* and setting their random indices of the bit array, we call this bit array the Bloom filter for the representation or the summary of the data set *S*. The random indices of bit array are called the insertion bits of the Bloom filter, and setting random indices of bit array is called insertion an element into a Bloom filter.

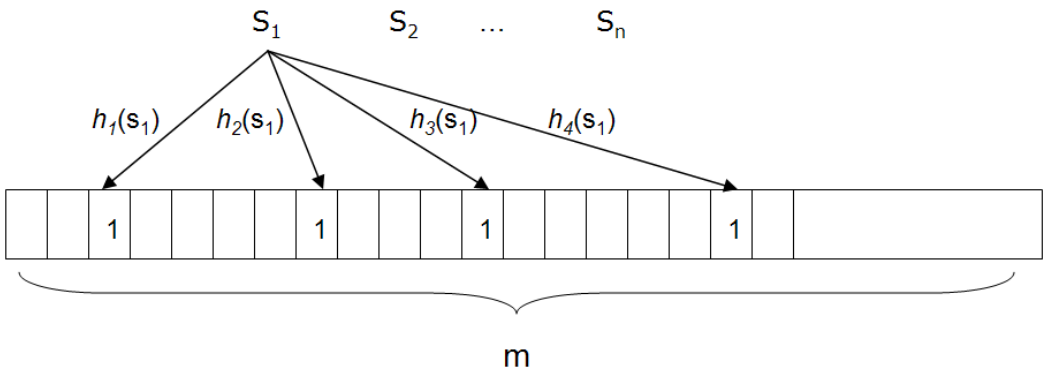


Fig. 2-1 Bloom filter data structure

Inserting an element into a Bloom filter is an operation to set *k* random indices on the bit array. The computation time of inserting operation is depended on the

computation of hash function, since the performance of hash functions is determined by the hashing algorithm. The hash function algorithms we use here are MD5 and SHA-256, which are digest message algorithms that produces 128 bits and 256 bits hashing values for arbitrary byte length of a message. In this thesis, the method of choosing a class of k independent hash functions from the 128 bits produced by MD5 is similar to the previous work [5] where k is smaller than eight. We divided the 128 bits digest message into eight 16-bit words for the number of hash functions k, and the each random index of an element is the modulus of different word by the size of Bloom filter m. If k was larger than eight, the first eight hash functions are come from MD5, and we further divide the 256 bits digest-message of SHA-256 into sixteen 16-bit words in the same way for the rest of hash functions k. We assume that the number of hash functions k in this thesis is smaller than 24, and the hash functions generated by MD5 and SHA-256 can satisfy the randomization property of our simulations.

To query whether an element is in a data set represented by a Bloom filter, we can check whether all the k indices of the bit array corresponding to the element are all true. Hence, we simply check the k indices produced by independent hash functions for the querying element, and we say this element is in the data set if the k indices of the bit array of this element were all true. When doing membership query, it will be not only to query a single element but also to compare more elements with Bloom filter. If there were two data sets S_1 and S_2, the comparison of two data sets S_1 and S_2 would be the indices checking of two bit arrays because two data sets have two bit arrays for their data set summaries. Instead of checking k indices for querying an element, we check m indices, where m is the size of Bloom filter in data set S_1.

The false positive of Bloom filter is an error that an element was not in the data set but Bloom filter reported it is. The probability of this error occurrence in Bloom

filter is called the false positive rate. In Fig. 2-2 there are two Bloom filters, one of which is the bit array B1 and the other is the bit array B2. B1 summarized the data set S_1= {{a}, {b}}, and B2 summarized the data set S_2= {{a}, {c}, {d}, {e}}. Note that the indices of B1 which were set to true are as same as the indices of B2, and then we can say the B1 is a subset of B2. Although the B1 is contained in B2, the S_1 is not a subset of S_2. The element {b} is not in S_2, but the Bloom filter reported it is. The situation is called the false positive of Bloom filter, and the correctness of Bloom filter is depended on the false positive rate, which is the probability of false positive occurrence. When doing set comparison, it is needed to generate two bit arrays to represent the separate data set and to check *m* indices. Moreover, we can do Boolean operation AND for two bit array for set intersection operation, and OR operation for set union.

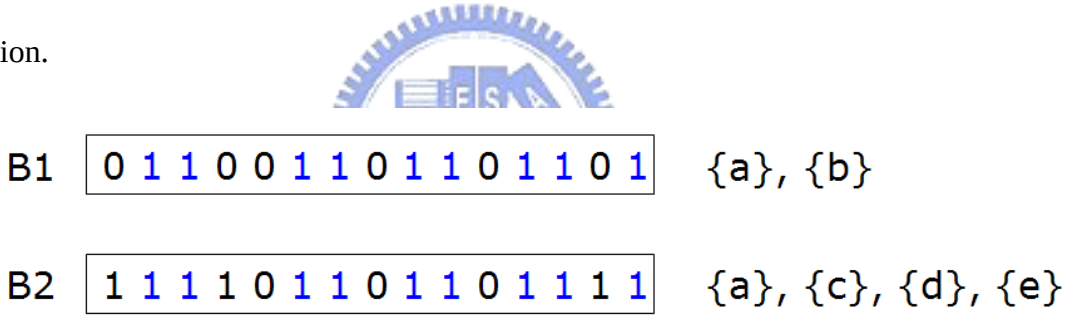


Fig. 2-2 Two Examples of Bloom filter

The correctness of Bloom filter is correlated with the size of Bloom filter *m*, the number of hash functions *k* and the number of the insertion elements of a data set *n*. The definition of the false positive has been described, and we defined the false positive rate is the error rate that an element is not in the data set but its random indices were true in the Bloom filter. We assumed that the *k* hash functions are all independent and perfect random, the size of Bloom filter is *m* and the number of insertion elements in data set is *n*. Before finding the false positive rate *f*, we calculate the probability *p*, which indicates a indices of Bloom filter was still 0 after inserting *n*

elements, is showed as Equation 2- 1.

$$p = (1 - \frac{1}{m})^{nk} \approx e^{-nk/m} \quad (2\text{-}1)$$

The probability p would be modified if the number of insertion bits $n*k$ was change. Based on Equation 2-1, the false positive rate can be thought that choosing k indices of a Bloom filter randomly, and all of k indices are true after inserting $n*k$ bits into Bloom filter. The false positive equation is expressed as follows.

$$f = (1\text{-}p)^k \approx (1 - e^{-nk/m})^k \quad (2\text{-}2)$$

After finding the false positive rate of Bloom filter in Equation 2-2, the relation between m, n and k needs to be derived for optimal configuration. Based on Equation 2-1 and Equation 2-2, the false positive rate f can be rewritten as $\exp k \ln(1-p)$. We let $g(k) = k \ln(1-p)$, and find differential value of $g(x)$ be 0 in order to find optimal value of k. In the differential equation of $\frac{dg}{dk} = \ln(1 - p) - \frac{\ln p}{(1 - p)} p$, the domain of p is $0 < p < 1$, and the p is 1/2. The optimal relation between m, n and k is expressed as follows.

$$p = 1/2 \text{ for } 0 < p < 1, \text{ then } e^{-nk/m} = 1/2 \ , \quad k = (m/n)\ln 2 \quad (2\text{-}3)$$

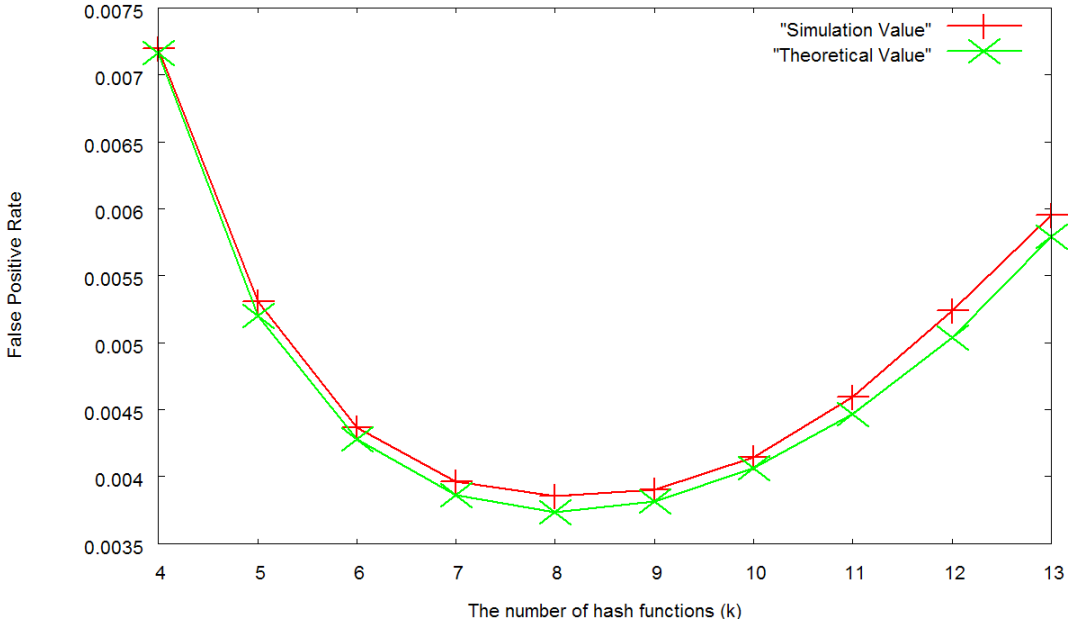


Fig. 2-3 The false positive rate using different k

In Equation 2-3, it is clear that an optimal value p for a Bloom filter is 1/2; moreover, by given any two of variables m, n and k, another variable can be determined. In other words, the ideal number of insertion bits $n*k$ in a Bloom filter would be equal to $m*ln2$. If the number of insertion bits is equal to $m*ln2$, the false positive f would be approximate to the value of $(1/2)^k$. In Fig. 2-3, the number of insertion elements n is 44, and the size of Bloom filter m is 512. We used the different numbers of hash functions k

from 4 to 13, and we find that the optimal k was 8. The simulation results are approximate to the theoretical value based on Equation 2-3 that optimal k is equals to $(m/n) * \ln 2$.

2.2 Numerical Range Query

In the distributed applications, a Bloom filter is used for membership query, and a range query is to query whether a numerical element is in the query range of the numerical attribute of a data set. For example, if there was a user who published a numerical element “15” with numerical attribute name “Age” and another user might want to query whether there has any number from “10” to “60” in “Age” attribute, the published Bloom filter was compared with the query Bloom filter. The query range which was represented by Bloom filter should contain the numerical elements from 10 to 60. The method to insert a numerical attribute into Bloom filter is to concatenate the data bytes of the numerical attribute name and the data bytes of each numerical element of this attribute, and then project each catenation onto the random indices of bit array. In other words, we first transform the attribute name and numerical elements into bytes, and then concatenate them into an inserted element of Bloom filter. For example, if we want to insert a numerical attribute whose name “Age”, and its query

range of “Age” attribute is from 10 to 60; therefore, the total 61 data byte concatenations from “Age” + 10 to “Age” + 60 will be projected onto the bit array. It might lead to that there are too many elements inserted into query Bloom filter. In the definition of Bloom filter, each element in a data set that was represented by Bloom filter will be projected onto the random indices of a bit array; Therefore, the probability *p* was getting small and the false positive rate *f* would be increasing. Since most of the range queries of Bloom filter are to query numbers, we consider a rage query of the Bloom filter in this thesis is a numerical range query. A numerical range R_2 $\{b_1, b_2 \ldots b_n\}$ is a subset of R_1 $\{d_1, d_2 \ldots d_m\}$ if $b_1 \geqq d_1$and $b_n \leqq d_m$ because both R_1 and R_2 are numerical rage, whose numerical elements are countable and continuous. Our assumption is that if there are two numerical elements e_1 and e_2 contained in a range *R*, the range R would contain all numerical elements from e_1 to e_2. If there was a range query for whether R_1 is contained in R_2, we query the numerical elements from b_1 to b_n about whether R_1 is contained in R_2. In Fig. 2-4, there are two numerical ranges R1 and R2, and the range query R1 to R2 is matching because each number of R2 is contained in R1.

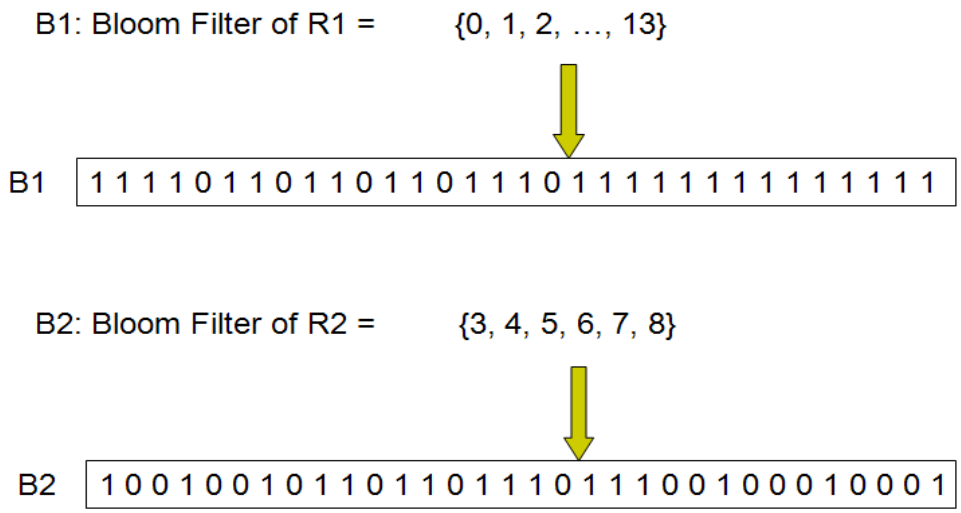
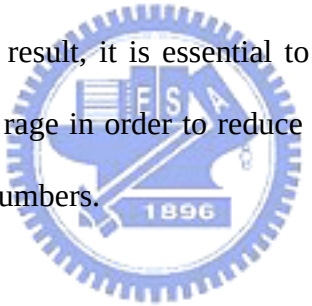


Fig. 2-4 Range query using Bloom filter

According to Equation 2-2, the number of insertion elements *n*, would increase

the false positive rate if there are too many continuous and numerical elements inserted into a Bloom filter. For efficient rage query, there are two Bloom filters to represent the query range and being query range, every numerical element of each range would be inserted into Bloom filter; therefore, if any number was an element or a subset in the query range, the range query is match when comparing the Bloom filters of query range and the Bloom filter of being query range. For more detail description as following: A query range would include many continuous numbers, and it then have many indices of its bit array to set 1; therefore, the number of insertion bits which were inserted its Bloom filter affects the false positive rate of the Bloom filter. Based on Equation 2-2, the numerical query range which has many continuous and countable elements will have many insertion bits and increase the false positive rate of the Bloom filter. As a result, it is essential to find more efficient method to represent the numerical query rage in order to reduce false positive rate if this query range which contained many numbers.



Chapter 3 Bloom Filter Design for Range Query

When the number of elements stored in a Bloom filter was larger than a threshold, the false positive rate increases dramatically. The design of Bloom filter is needed to be modified in order to store a large range of continuous numeric numbers. As we have mentioned before, inserting all elements of a data set into Bloom filter by using a set of hash functions. In most situations, the size of Bloom filter *m* and the number of hash functions *k* are predefined when inserting elements into Bloom filter or doing Bloom filter comparisons. As a result, the number of insertion elements is the important factor of the false positive rate of Bloom filter. In order to reduce the false positive rate of Bloom filter when inserting the query range of numerical attribute, three schemes are proposed to remedy the original method when too many numbers of query range inserted into a Bloom filter. The focus of our schemes is to compress the number of insertion bits when too many numerical elements were inserted into Bloom filter. Moreover, we try to find the optimum configuration of our schemes for the minimum false positive probability.

We assume that the range query of a numerical attributes in Bloom filter had its native minimum and maximum numbers in the domain of this attribute. For example, the numbers of “Age” numerical attribute would distribute over 1~120, and the numerical elements of “Year” attribute would be distribute over 0~3000; therefore, it is necessary to considerate not only how many numbers is in the query range of the numerical attribute but also the numerical property of this attribute. For different domain of different numerical attributes, the optimal parameters of our proposed schemes would be different.

3.1 Division Scheme

For numerical range query, a simple way to reduce the false positive rate is to decrease the number of insertion bits inserted into Bloom filter. The scheme "Division" is to group the continuous numbers into different divisions, and the numbers in the same division are projected onto the same random indices of Bloom filter. The Division scheme extends the original insertion scheme and inserts continuous numbers of different division into bloom filter. In previous Bloom filter, every number in query range are projected onto a set of random indices of bit array, and the large number of insertion bits will decrease the probability *p* of Equation 2-1 when inserting too many numerical elements into Bloom filter. Instead of hashing every number and inserting them into Bloom filter, we simply group continuous number into divisions and only insert the first number of different divisions into Bloom filter. The Division scheme is to use several divisions to represent the continuous numbers of the query range of a numerical attribute, and the number of how many continuous numerical elements were grouped into a division is called "dividing-range". In Fig. 3-1, there was a numerical attribute whose numerical elements were inserted into a Bloom filter by Division scheme, and the dividing-range of Division scheme was 5. For example, if there was a numerical attribute which contained the numbers from 2 to 13, the Division scheme only inserted the number {0}, {5} and {10} numerical elements into a Bloom filter rather than inserted the 12 numbers from 2 to 13 into a Bloom filter. Due to the number of the insertion numbers *n* was decreased by Division scheme, the number of insertion bits *n*k* was also reduced, and then the false positive rate was decreased.

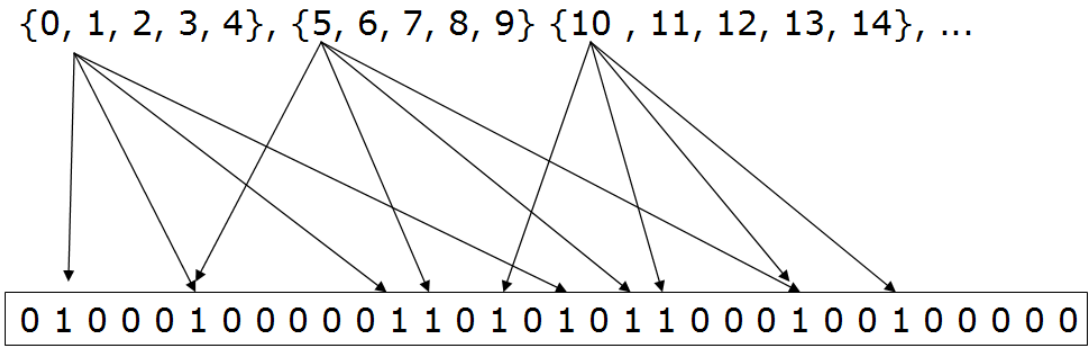


Fig. 3-1 Group continuous numbers into divisions

Although the number of insertion bits was decreased, an additional penalty for Division would increase the error rate. The penalty of this scheme is the effect of dividing range factor, which incurs the additional false positive for two different numbers being regarded as the same. In previous example of Fig. 3-1, the dividing-range was 5, and if the domain range of a number was from 0 to 1000; therefore there were about 200 divisions in this number range. We assumed the probability distribution of the selection of a number was uniform distribution, and there were two numbers selected randomly. We could simply say that the error probability of two numbers seeming to the same was about 1/200. Because we grouped the continuous numbers into divisions, the number of insertion element would be change. The native false positive rate f_d of using Division scheme would be different to the false positive rate f of the original scheme. Because the number of insertion bits $n*k$ was changed in Division scheme, we first calculated the function of the number of insertion bits $g(n, d, k)$, where d was the value of the dividing-range of Division scheme. Because we knew the size of the query range n of a numerical attribute, but the start number of this query range was not certainly assured; therefore, we used the expectation value of how many division were used for this query range, and the $g(n, d, k)$ was the expectation value of the number of insertion bits of this query range. Finally, the number of insertion bits can be written as follows.

$$g(n,d,k) = k\sum_{i=0}^{d-1}\lceil (n+i)/d \rceil / d \quad (3\text{-}1)$$

From Equation 3-1, the probability p of Division scheme is re-written as follows.

$$p = (1-\frac{1}{m})^{g(n,d,k)} = e^{-g(n,d,k)/m} \quad (3\text{-}2)$$

After calculating the probability p that a index of the bit array was still 0 after inserting $g(n, d, k)$ bits into a Bloom filter, the native false positive rate f_d of using Division scheme is written as follows.

$$f_d = (1-p)^k = (1-\mathrm{e}^{-g(n,d,k)/m})^k \quad (3\text{-}3)$$

Although the false positive rate f_d of Division scheme was calculated, the penalty of Division scheme was still needed to be considered. As we have mentioned before, Division scheme is to group the continuous numbers into different divisions in the dividing-range d. The probability that two different numbers were regard as the same is the value of dividing-range d divided by the domain R of a numerical attribute. In Division scheme, the error that a number was not in the query range but in the division of the query range is called “dividing error”. In other words, the total false positives of Division scheme include not only the native false positive but also the dividing error events. The number of dividing error events is still the expectation value because that we did not know the start number of the query range. Because the numbers of a query range in our assumption were all continuous and countable, the dividing error events of a query range can be determined if the start number of the query range and its size had been decided. Therefore, we need to consider all case that the start number’s position in its division, and we then calculated the expectation value of the dividing error events if the dividing-range had been decided. The expectation value of dividing error events is written as follows.

$$E(\textit{Dividing Error events}) = \frac{\sum_{i=0}^{d-1} i + (n+i) \bmod d}{d} = \frac{d(d-1)}{d} = d-1 \qquad (3\text{-}4)$$

After calculating the native false positive rate f_d of Division scheme and the expectation value of dividing error events, the total false positive rate f of Division scheme can be rewritten as follows.

$$f(m,k,R,n,d) == \frac{R-n-(d-1)}{R-n} f_d + \frac{d-1}{R-n} = \frac{R-n-(d-1)}{R-n}(1-p)^k + \frac{d-1}{R-n} \qquad (3\text{-}5)$$

In Equation 3-3 $f_d = (1 - e^{-g(n,d,k)/m})^k$, the native false positive rate of Division scheme is different to the Equation 2-2 $f = (1 - e^{-nk/m})^k$ because the number of insertion bits in Equation 3-1 would be small than original insertion bits when d was larger than 1; as a consequence, the native false positive value of Equation 3-3 is less than the value of Equation 2-2. But we still need to considerate the error that two numbers are in the same division but they are not the same. For previous example, the range {2, 3…, 13} was inserted into Bloom filter with dividing range d = 5, and the number 14 was not in the range but Bloom filter reported it is. Such error is called "dividing error", and the Equation 3-4 shows the expecting value of dividing error events. In Equation 3-4, if the dividing range d is 1, the number of dividing error events will be 0; every number in the query range is projected onto its independent random indices; therefore, there is no dividing error, and the Division scheme with dividing-range d = 1 is as same as the original scheme.

In Fig. 3-2, there was a query range of a number attribute was inserted into a Bloom filter whose size m is 512 and the number of hash functions k is 8, and the domain of the numerical attribute R is 1000. The number of insertion elements of the query range n is 100. The variation in x-axis is the dividing-range d from 1 to 10. The total false positive rate of Division scheme based on Equation 3-5 is consistent with the simulation value depicted in Fig. 3-2, whose simulation environment which we

used here was single numerical attribute in single Bloom filter. The numerical attribute name in the simulation is “Range”. In Algorithm 3-1, the simulation is to randomly choose a length n range and a number within domain R with the same attribute name, and then based on the test of 10^6 random elements, we calculate the false positive rate by the comparisons of two Bloom filters BF1 and BF2. In our simulation the range and the number are selected in uniform distribution. Result of Fig 3-2 showed that the optimal parameter d is 3 or 4. Because the error rate will be linearly increasing when the number of insertion bits $g(n, d, k)$ was so small, the native false positive f_d of checking k indices can be ignored; however, the dividing error is increasing. As a consequence, the Division scheme can reduce the number of insertion bits and the native false positive rate f_d, but the dividing error is increasing if the dividing-range d was getting large.

Sumualtion Algorithm
BF1 : query range Bloom filter, BF2 : Comparison Bloom filter
fp : false positive, tn : ture nagative
1. fp $\leftarrow$ 0; tn $\leftarrow$ 0
2. for i form 1 to 1000000
3. if i mod 1000 = 1
4. clear(BF1) // reset all bits of BF1
5. generate a length n query range Q and inset it into BF1
6. end if
7. clear(BF2)
8. generate a random number e withen domain R and insert it into BF2
9. if BF2 $\subseteq$ BF1 and e $\notin$ Q
10. fp $\leftarrow$ fp + 1
11. elsif BF2 $\not\subset$ BF1 and e $\notin$ Q
12. tn $\leftarrow$ tn + 1
13. end loop
14. return fp/(fp + tn) // the false positive rate

Algorithm 3-1 Simulation Algorithm

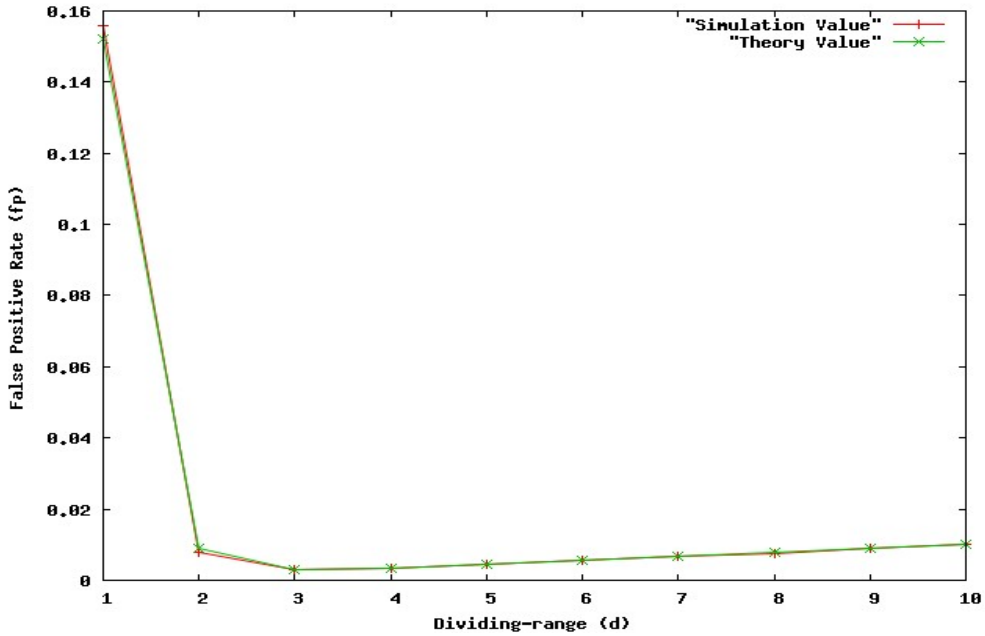


Fig. 3-2 The false positive rate of Division scheme

3.2 Overlapping Scheme

The “Overlapping” scheme like Division scheme uses less random indices of bit array than original scheme to represent the continuous numbers of a query range, and it eliminates the dividing range error caused by Division scheme. The Overlapping scheme inserts every number of a query range into Bloom rather than only the first number of a division. The Overlapping scheme uses a specialized class of projection functions H_s to insert elements into Bloom filter, and the random indices of the continuous numbers would be only different “shift-bit” s indices and same o indices between neighbor numbers where $o = k - s$. In Fig. 3-3, Instead of inserting only one dividing number {0} to represent 5 numbers {0, 1, 2, 3, 4} depicted in Fig. 3-1, the Overlapping scheme inserted all 5 numbers into Bloom filter; therefore, there is no dividing error in it. The second random index $H_1(1)$ of number {0} is the same as the first random index $H_1(1)$ of number {1}, and the second random index $H_1(2)$ of number {1} is the same as the first random index $H_1(1)$ of number {2}. By given the k of a Bloom filter and shift-bit s for Overlapping scheme, the k random indices of a

number *e* is written as follows.

$$\sum_{i=0}^{i<\lceil k/s \rceil-1}\sum_{j=1}^{j\le s} H_j(e+i)+\sum_{j=1}^{k\%s} H_j(e+(k/s)); \quad \text{for } s<k, k\%s>0$$
$$\sum_{i=0}^{i<k/s}\sum_{j=1}^{j\le s} H_j(e+i); \quad \text{for } s<k, k\%s=0 \qquad (3\text{-}6)$$

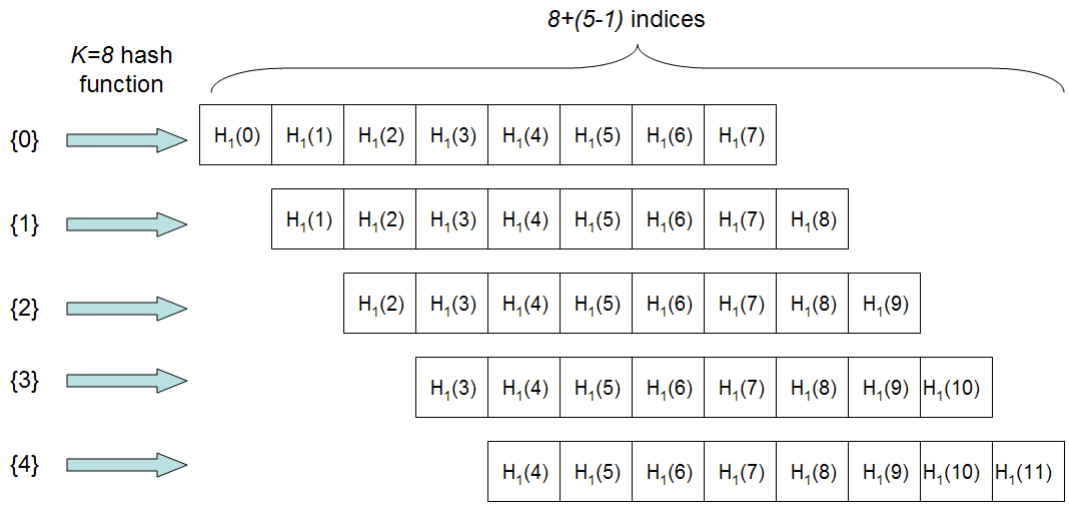


Fig. 3-3 Overlapping scheme with parameter *s* = 1

Although the Overlapping scheme can decrease the number of insertion bits, it still has the penalty of the random indices overlapping. If there are *n* continuous numbers in a query range, and we use "Overlapping" scheme with shift-bit *s* = 1 to insert *n* numbers into Bloom filter. Hence, instead of *n*k* random indices, there are only *k* + (*n*-1) random indices on the Bloom filter. According to Equation 2-2, the false positive rate is the probability that an element was not in Bloom filter but the *k* random indices of the bit array are true. In other words, an element is not in the data set but Bloom filter reported it is after checking *k* random indices of bit array. Because of the random indices overlapping in Overlapping scheme, some numbers in the numerical attribute have less number to checking indices. In Fig. 3-3, we assume the number of hash function *k* is 8, and the numerical element {0} and {1} are different in only one random index, and the {0} and {3} are different in only two. As a result, it is necessary to modify the original false positive rate equation in Equation 2-2. If there

was a query range from e_1 to e_2, the numbers whose indices overlapped with this query range is 2r, where r can be written as follows

$$r = \lceil k / s \rceil - 1 \tag{3-7}$$

To calculate the false positive rate f of Overlapping scheme, we need to recalculate the number of insertion bits. The number of insertion bits of using Overlapping is definite value rather than the insertion bits of using Division, and the function $q(n, s, k)$ is written as follows.

$$q(n, s, k) = (n-1)s + k \tag{3-8}$$

From Equation 3-8, the probability p of using Overlapping can be written as follows.

$$p = (1 - \frac{1}{m})^{q(n,s,k)} = e^{-q(n,s,k)/m} \tag{3-9}$$

From Equation 3-7 and Equation 3-9 the false positive rate f can be rewritten as follows.

$$f(m, k, R, n, s) = \frac{R-n-2r}{R\text{-}n}(1-p)^k + \frac{2}{R-n}\sum_{i=1}^{i<=r}(1-p)^{is} \tag{3-10}$$

In Equation 3-10, the false positive rate of Overlapping scheme is calculated in two conditions, one of which is no random indices overlapping and the other is with random indices overlapping. The penalty of Overlapping depends on the number of how many number having indices overlapping and the probability to selecting them from the domain of the numerical attribute. In Fig. 3-4, we inserted a numerical attribute whose domain R is 1000 and continuous numbers n of the query range is 100 inserted into a Bloom filter with m = 512 and k = 8. The simulation algorithm is as same as Fig. 3-2, and the false positive rate values of Equation 3-10 are approximate to the simulation results. It is apparent to see that the relationship between the shift-bit s and false positive is exponential.

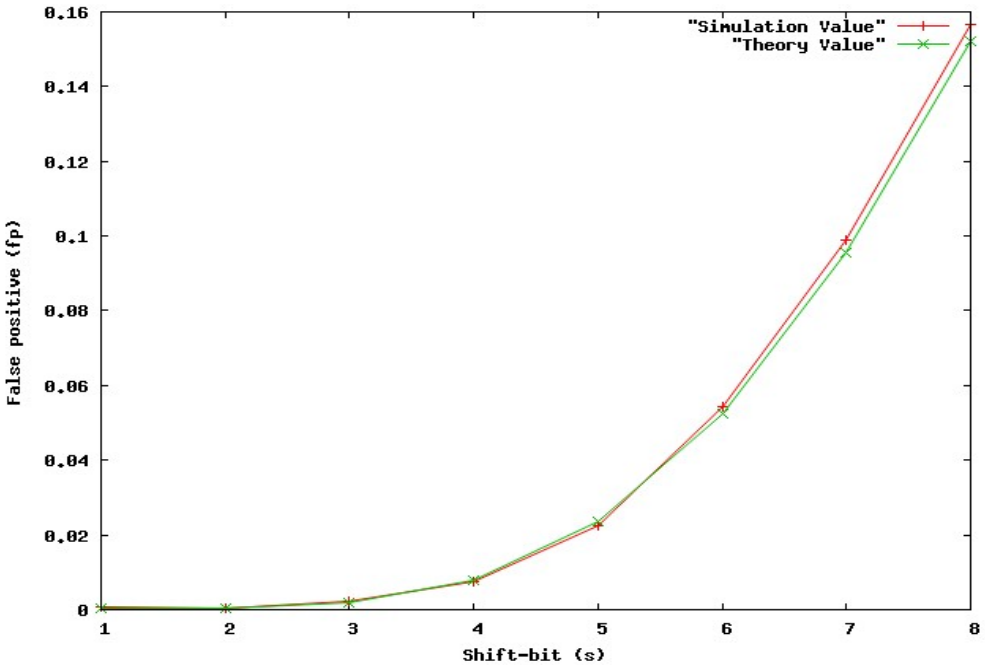


Fig. 3-4 The false positive rate of Overlapping Scheme

3.3 Division-Overlapping Scheme

The “Division-Overlapping” scheme is the combination of two precious schemes the Division and the Overlapping. It not only defines dividing-range *d* to make continuous numbers to different divisions, but also uses shift-bit *s* to indices overlaps between the indices of continuous numbers. For the Division scheme, the Overlapping scheme could make compensation for the dividing error caused by the parameter dividing-range of Division scheme; for Overlapping scheme, the Division scheme could help Overlapping scheme make more compression of random indices. In Fig. 3-5, the continuous numbers from 0 to 24 was represented by 12 random indices, and the two parameters dividing-range *d* and shift-bit *s* of Division-Overlapping scheme are 5 and 1. This scheme first grouped five continuous numbers into the same division, and it made random indices overlapping between neighbor divisions, and there was only one different random index between neighbor divisions.

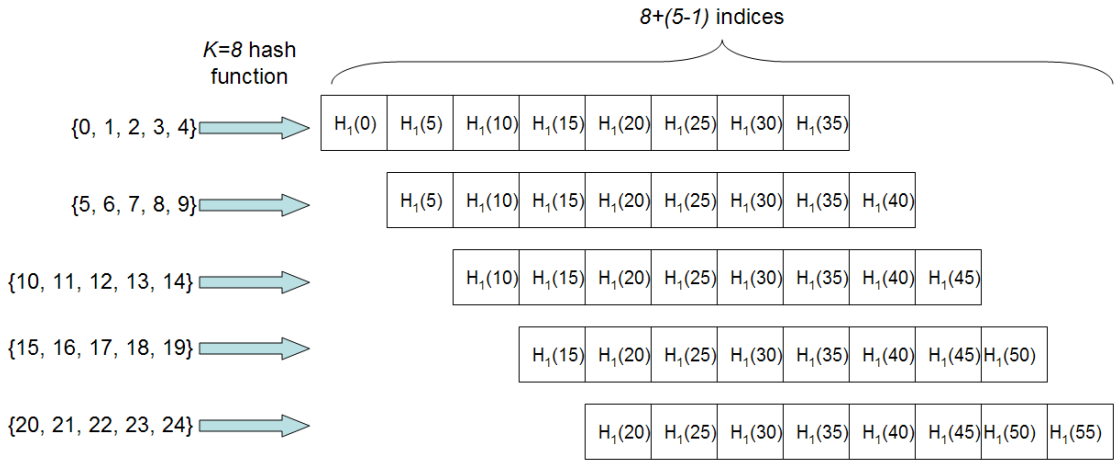


Fig. 3-5 Division Overlapping scheme with $s = 1$ and $d = 5$

The number of insertion bits of Bloom filer is our main concern for our schemes. The Division-Overlapping scheme can not only make the number of insertion bits less than "Overlapping" but also have less dividing error events than Division in the same number of insertion bits. Moreover, the Division-Overlapping scheme has more flexibility in adjusting the parameter (d, s) rather than only one parameter d of Division and s of Overlapping. The Division-Overlapping is as same as Division if the parameter $s = k$ and $d > 1$, and it is as same as Overlapping if the parameter $d = 1$ and $0 < s < k$.

The false positive rate of the Division-Overlapping scheme is still the combination of two schemes Division and Overlapping, and the theoretical value can be calculated by previous equations of Division and Overlapping. To calculate the false positive rate f, we first calculate the number of insertion bits in Bloom filter. Because the Division-Overlapping first grouped the continuous numbers into different divisions in dividing-range d for the Division scheme, the number of insertion bits is the expectation value. It then used shift-bit s to make random indices overlapping between neighbor divisions. Finally, the equation $w(n, d, s, k)$ to calculate the number of insertion bits can be written as follows.

$$w(n,d,s,k) = (\sum_{i=0}^{d-1} \lceil (n+i)/d \rceil / d - 1)s + k \quad (3\text{-}11)$$

After calculating the number of insertion bits, the probability *p* that a bit of the Bloom filter was still 0 after inserting *w*(*n*, *d*, *s*, *k*) bits can be written as follow.

$$p = (1 - \frac{1}{m})^{w(n,d,s,k)} = e^{-w(n,d,s,k)/m} \quad (3\text{-}12)$$

From previous Equation 3-5 and Equation 3-10 the false positive rate *f* can be written as follows.

$$f(m,k,R,n,d,s) = \frac{R-n-(d-1)-2rd}{R-n}(1-p)^k + \frac{d-1}{R-n} + \frac{2d}{R-n}\sum_{i=1}^{r}(1-p)^{is} \quad (3\text{-}13)$$

In Fig. 3-6 and 3-7, we inserted a numerical attribute whose domain *R* is 10000 and continuous numbers *n* of a query range is 1000 inserted into a Bloom filter with *m* = 512 and *k* = 8. The theoretical value of Equation 3-13 is consistent with the simulation value. The differences from previous simulation environment are the domain *R* of numerical attribute and the number of insertion elements *n* because more insertion elements could have better discrimination for the comparison of theory values and simulation results; the variation of x-axis was parameter *s* from 1 to 8, and the variation of y-axis was parameter *d* from 1 to 500.

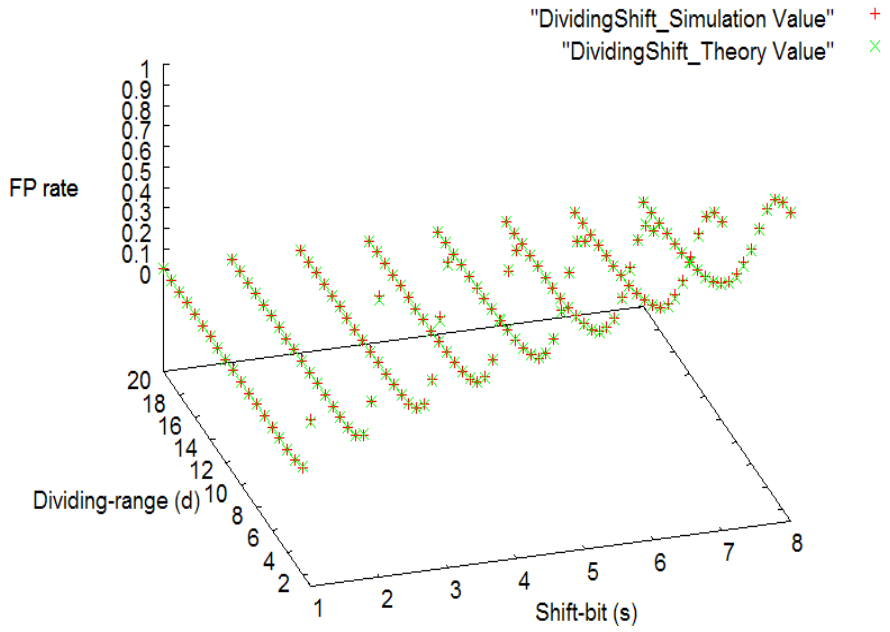


Fig. 3-6 The false positive rate of Division-Overlapping Scheme

Fig. 3-6 showed a part of theory and simulation results, whose the shift-bit *s* was from 1 to 8 and the dividing-range d is from 1 to 20; Fig. 3-7 showed the other results, whose the shift-bit *s* was from 1 to 8 and the dividing-range d is from 21 to 500.

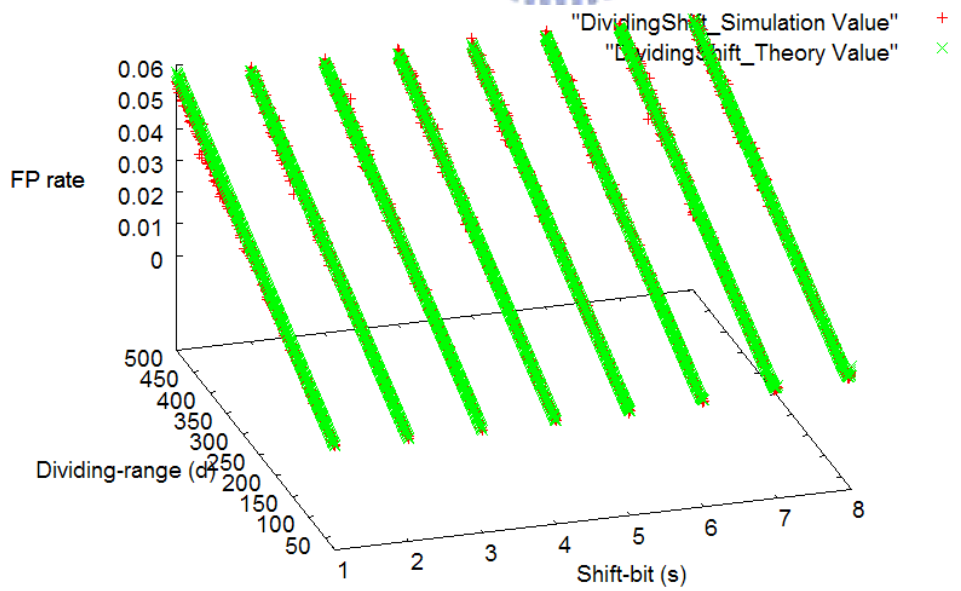


Fig. 3-7 The false positive rate of Division-Overlapping Scheme

In Fig. 3-6, it is apparent to see that the relationship between shift-bit *s*,

dividing-range *d* and false positive rate (FP) is exponential when the dividing-range *d* is approximate to 1 and shift-bit *s* is from k to 1 because the number of insertion bit is too large to be the main factor of the false positive rate. In Fig. 3-7, the relationship between *d* and false positive is linear because that the insertion bits compression can make the number of insertion bits very small so that the dividing error becomes the main factor of the false positive rate. The simulation result is also consistent with previous simulation results depicted in Fig. 3-2 and Fig. 3-4.

3.4 Optimal Parameters for Bloom Filter

In previous three sections the Division, the Overlapping and the combination of both Division-Overlapping were proposed to make insertion bits compression for reducing the false positive rate. The original parameter optimization for *m*, *n* and *k* in Equation 2-2 would be not suitable for our insertion bits compression schemes because of the additional parameters dividing-range *d* and shift-bit *s* affecting the false positive rate of a Bloom filter. We consider two situations, one of which is that single numerical attribute in a Bloom filter and the other is that multiple numerical attributes in a Bloom filter, and we try to find the optimal parameters *d* and *s*. The methods to find their optimal parameter will be proposed. In the case of single numerical attribute in a Bloom filter, the optimal parameters for three compression schemes will discussed; however, we will only discuss Division-Overlapping scheme in the case of multiple numerical attributes in a Bloom filter because Division-Overlapping can be the Division scheme if $s = k$ and $d > 1$or the Overlapping scheme if $d = 1$ and $0 < s < k$.

3.4.1 Single Numerical Attribute in a Bloom filter

A."Division" scheme

The optimal parameter dividing-range *d* is decided by the insertion numbers *n* and the domain range *R* of the numerical attribute when the other parameters including *m*, *k* and range *n* have been predefined. According to the previous simulation result depicted in Fig. 3-2, we found that when the dividing-range *d* increased, the false positive rate would first get less and less; but after one point, the false positive rate would be increasing linearly. The optimal dividing-range *d* for the "Division" scheme is the x-axis value on the point, which is also the inflection point on the false positive rate curve.

In Fig. 3-8, the line is the searching path, and if the value of false positive rate on the point $d = 4$ was larger than the value on previous point $d = 3$, the optimization stops and decides that the previous point is the optimal *d* for the Division scheme. In this figure, the optimal dividing-range *d* is 3 for the range $n = 100$ in the domain $R = 1000$, and the false positive rate on the $d = 3$ is about 0.003. Our scheme to find the optimal dividing-range *d* is to find the inflection point based on the false positive rate of Equation 3-5 when the parameters m, n and k were predefined. The false positive rate is decreasing exponentially before the inflection point because that the number of insertion bits was effectively compressed; but after the inflection point, the error rate is increasing linearly because of the penalty of dividing-range *d* which grouped *d* continuous numbers into a divisions. Therefore, there is only local extreme minimal value, which is the global minimal value on the curve, and the optimal dividing-range *d* is the inflection point on the false positive curve.

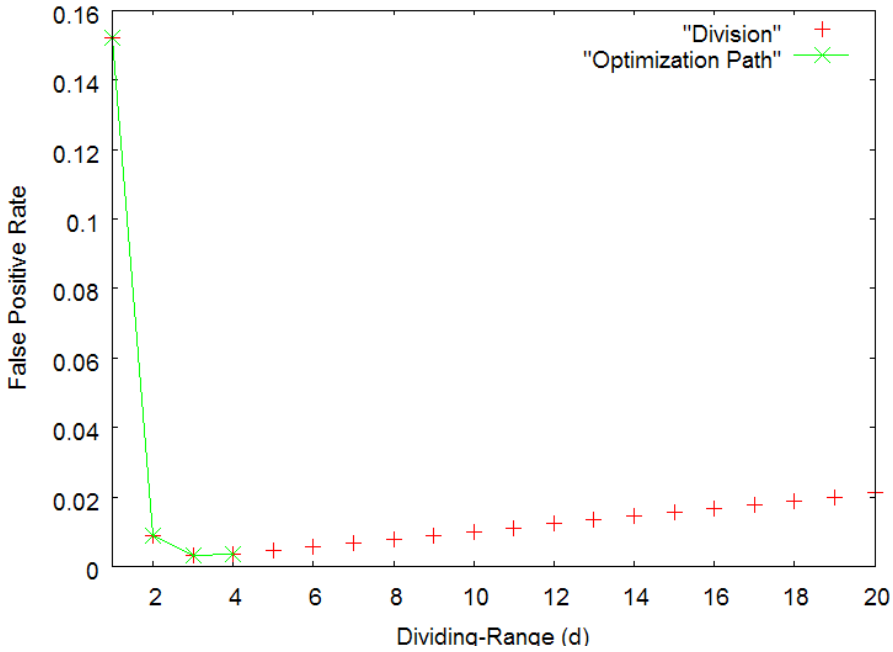


Fig. 3-8 Searching the inflection point of false positive rate

B. “Overlapping” scheme

Finding optimal parameter shift-bit *s* is as same as Division when the size of Bloom filter *m*, the number of insertion numbers *n* and the number of hash functions *k* were predefined. The difference of finding optimal parameter between Overlapping and Division is that the variation of shift-bit *s* is from 1 to *k*, but dividing-range *d* is from 1 to *n*. The way to find the inflection point on the false positive curve is still applied to search optimal shift-bit *s* for the Overlapping scheme. In Fig. 3-9, the number of insertion element *n* is 40 rather than 100, and the other parameters are as same as Fig. 3-8. We find that the point $s = 4$ on the false positive rate curve was larger than the point $s = 3$ on the curve, so that the point $s = 3$ is the optimal shift-bit. Because the penalty of the Overlapping scheme is the random indices checking error, and we found the curve is always concave upward for the parameters *m*, *n*, *k* and *s* are all larger than 0. Therefore, the property of Overlapping that there is only one extreme value on the curve is as same as Division scheme.

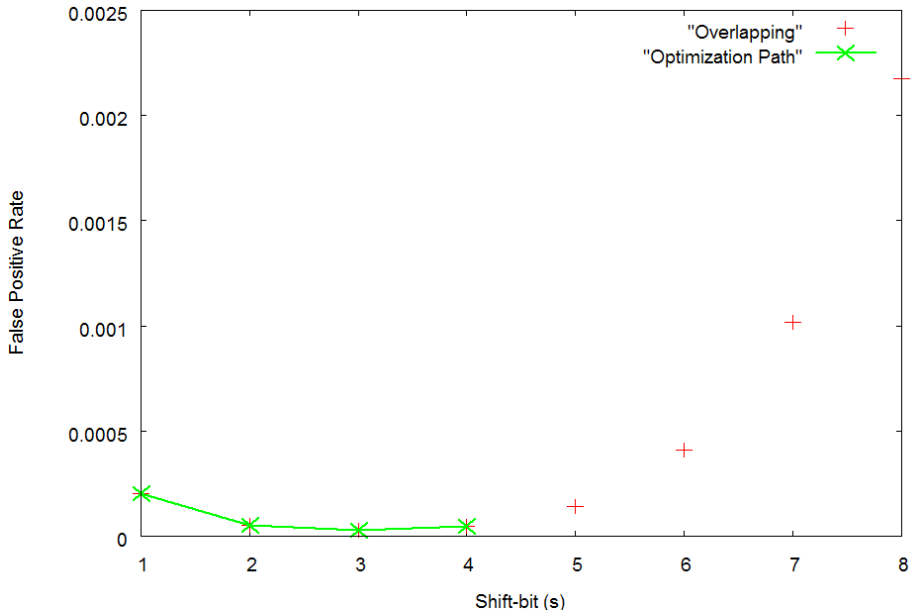


Fig. 3-9 Searching the inflection point of false positive rate

C. “Division-Overlapping”

Finding the optimal parameter (*d*, *s*) dividing-range *d* and shift-range *s* is similar to previous two optimization search, and the search path is 2-dimentional.In Fig. 3-10, there was a query range whose the number of continuous numbers *n* is 1000 inserted into a Bloom filter, and the domain of the numerical attribute *R* was 10000; the size of the Bloom filter *m* was 512, and the number of hash functions *k* was 8. The optimization searching first searched on the curve (*d*, 1) where the variation is dividing-range *d*, and the shift-bit *s* is constant 1. The method to find the inflection point (d_1, 1) on the curve (*d*, 1) is as same as previous searching method in Division. Next, we find the inflection point (d_2, 2) on the curve (*d*, 2). If the value of inflection point (d_2, 2) was larger than the inflection point on (d_1, 1), the searching path stops and decides that (d_1, 1) is the optimal parameters for the Division-Overlapping. In this figure, the optimal parameter (*d*, *s*) for the Bloom filter is (6, 1). The Algorithm 3-1 “Optimization-SingleAttribute” is to find the optimal parameters dividing-range *d* and shift-bit *s* for the numerical attribute with *n* continuous numbers and the domain range

R in a Bloom filter. The time complexity of finding optimal parameter (*d*, *s*) for a numerical attribute is $O(n \times k)$. This algorithm can also be applied to find the optimal parameter for Division or Overlapping scheme if we made constraint on choosing parameter dividing-range *d* and shift-bit *s* in the optimization search.

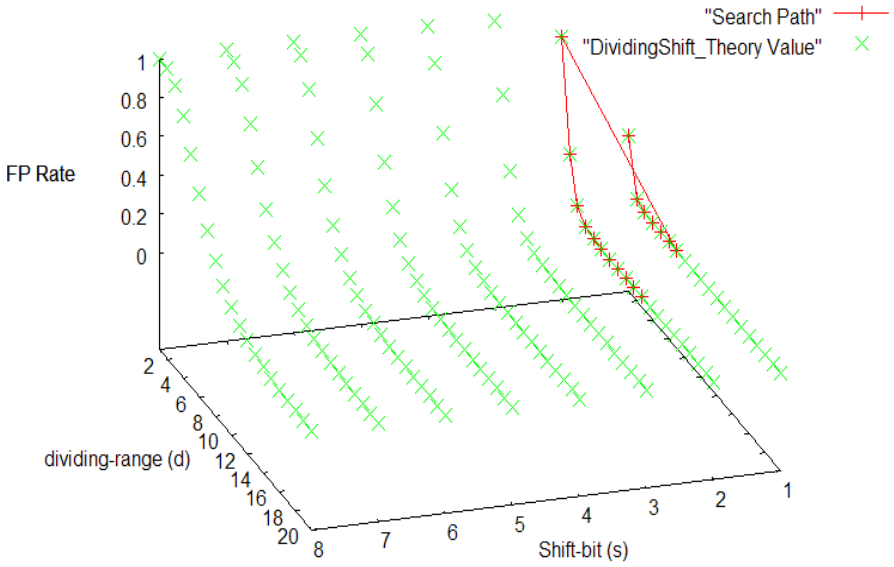


Fig. 3-10 Searching the inflection point of two-dimensional false positive curve

3.4.2 Multiple Numerical Attribute in a Bloom filter

In the case of multiple numerical attributes in a Bloom filter, the target is to find the optimal parameter (d_i, s_i) of Division-Overlapping scheme for each numerical attribute A_i, so that the average false positive rate of all attributes would be minimal. The false positive rate of each numerical attributes A_i depends on the number of total insertion bits and its parameters (d_i, s_i) for Division-Overlapping scheme. Because there are more than one numerical attributes in the data set, the number of total insertion bits is the sum of the number of insertion bits of each numerical attributes. Therefore, we must consider all possible of each parameter (d_i, s_i). Based on Equation 3-11and Equation 3-13, we modify the number of insertion bits equation and the false positive rate equations in Division-Overlapping scheme for multiple numerical

attributes parameters optimization. In Equations 3-14, the parameter “addBits” is the number of additional insertion bits, which is the sum of total insertion bits of the other attributes.

$$w'(n,d,s,k,\text{addBits}) = w(n,d,s,k) + \text{addBits} \quad (3\text{-}14)$$

With the actual insertion bits in Bloom filter, the false positive rate of each numerical attribute in that same Bloom filter can be calculated as follows.

$$p = (1-\frac{1}{m})^{w'(n,d,s,k,\text{addBits})} = e^{-w'(n,d,s,k,\text{addBits})/m}$$

$$f'(m,k,R_i,n_i,d_i,s_i,\text{addBits}) = \frac{R_i - n_i - (d_i - 1) - 2r_i d_i}{R_i - n_i}(1-p)^k + \frac{d_i - 1}{R_i - n_i} + \frac{2d_i}{R_i - n_i}\sum_{i=1}^{r}(1-p)^{is}$$

(3-15)

TotalFalsePositiveRate(m, k, S, FP)
S : { A_1, A_2 ... } the data set containing numerical attributes
A_i : { n_i, R_i, d_i, s_i } :
n_i : the number of insertion elements, R_i : the domain of this numerical attribute
d_i : the parameter dividing - range of "Division - Overlapping"
s_i : the parameter shift - bit of "Division - Overlapping"
FP : the array containing each false positive rate of attribute A_i in S

1. totalBits = $\sum_{A_i \in S} w(n_i,d_i,s_i,k)$ // the total insertion bits in Bloom filter
2. TotalFP ← 0
3. for i ← 1 to sizeof(S)
4. FP[i] ← $f'(m,k,R_i,n_i,d_i,s_i,\text{totalBits} - w(n_i,d_i,s_i,k))$
   // recalculate the false positive rate of each A_i
5. TotalFP ← TotalFP + FP[i]
6. end loop
7. return TotalFP // return the sum of false positive rates of all attributes A_i

Function 3- 1 Calculate total false positive

```
OptiSingleAttribute(m, k, A_i, S) // used by multiple attribute optimization
1. for i ← 1 to k   // change shift - bit
2.     fpTemp1 ← 1
3.     for j ← 1 to A_i   // change dividing - range
4.         set s_i ← i
5.         set d_i ← j
6.         fpTemp2 ← TotalFalsePositiveRate(m, k, S, FP)
           // calculate the total false positive rates of S
7.         if fpTemp2 < fpTemp1 then
8.             fpTemp1 ← fpTemp2
9.             DividingTemp1 ← j
10.        else // find the inflection point on the curve
11.            break
12.        end if
13.    end loop
14.    if fpTemp1 < fpMin then
15.        shifting ← i
16.        dividing ← DividingTemp1
17.    else // find the inflection point on the curve
18.        set s_i ← shifting
19.        set d_i ← dividing
20.        break
21.    end if
22. end loop
```

Function 3- 2 Finding the optimal parameter (d_i, s_i) of each numerical attribute

There are two functions, one of which is the function “TotalFalsePostiveRate” depicted in Function 3-1, and the other of which is the function “OptiSingleAttribute” depicted in Function 3-2. They are used by our parameter optimization algorithm “Optimization-MultipleAttribute” depicted in Algorithm 3-2. In Function 3-1, the false positive rate of each attribute A_i with the same parameters (m, k) is calculate based on each parameters (n_i, R_i, d_i, s_i); after calculation of the false positive rate of each attribute A_i, the total false positive rate of the summation of all numerical attributes would be return. Because the probability of each attribute to be queried is

equal, the minimal average false positive rate of all attributes is the target of our optimization algorithm. Instead of using average false positive rate, we use the total false positive rates of all numerical attribute for simplicity. In Function 3-2, we modified the previous Algorithm 3-1 in line 4 to the calling Function 3-1 because our concern is the total false positive rate of the summation of all attributes rather than individual false positive rate. After calculation of Function 3-2, the parameter (d_i, s_i) of the numerical attribute A_i would be optimal for the total false rate.

There are two phases in our multiple numerical attributes parameter optimization algorithm. In the first phase "Preprocess" of Algorithm 3-2, we first assumed that there was only one numerical attribute in a Bloom filter, and then found optimal parameter (d_i, s_i) and the false positive rate of each numerical attribute was calculated by Algorithm 3-1. After the single attribute optimization, the false positive rate of each numerical attribute and the number of insertion bits of each numerical attribute will be calculated again where all attribute are inserted the same Bloom filter. The total false positive rate of all attributes in the Bloom filter and the false positive rate of each numerical attribute are used to next phase. The purpose of first phase is to calculate the false positive rate FP[i] of each attribute A_i in the data set S.

In the second phase "Iteration", we optimize the parameter (d_i, s_i) of each numerical attribute. The numerical attribute with larger false positive rate is first because the numerical attribute with larger false positive rate may have more insertion elements than the other attributes. Therefore, we first optimized the parameter of this attribute and calculated the false positive rate again in line from 19 to 21 of Algorithm 3- 2. In line 23 of Algorithm 3-2, the termination condition is that the value of total false positive rate did not change after this iteration. Instead of using brute-force search for all possible parameters combination of all numerical attribute , the time complexity of finding the optimal total false positive rate in our optimization

algorithm is about O($l \times n \times k$) rather than O($(n \times k)^l$), where l is the number of numerical attributes in the data set S, and n is the size of the query range which was larger than the others. The purpose of second phase is to find individual optimal parameter (d_i, s_i) of the Division-Overlapping scheme for each numerical attribute A_i of data set S to have minimal average false positive rate in multiple numerical attribute.

```
Optimization - SingleAttribute(m, k, R, n)
m : the size of Bloom filter, k : the number of hash functions, R : the domain of numerical attribute
n : how many continuous numbers
1. for i ← 1 to k   // change shift - bit
2.     fpTemp1 ← 1
3.     for j ← 1 to n  // change dividing - range
4.         fpTemp2 ← f(m, k, R, n, j, i) // where f is the Equation 3 - 8
5.         if fpTemp2 < fpTemp1 then
6.             fpTemp1 ← fpTemp2
7.             DividingTemp1 ← j
8.         else // find the inflection point on the curve
9.             break
10.        end if
11.    end loop
12.    if fpTemp1 < fpMin then
13         fpMin ← fpTemp1
14.        shifting ← i
15.        dividing ← DividingTemp1
16.    else // find the inflection point on the curve
17.        return dividing and shifting // return the optimal parameter < d, s >
18.    end if
19. end loop
```

Algorithm 3-2 Optimization-SingleAttribute

Optimization - MultipleAttribute(m, k, S, FP)

// Phase 1 Preprocess

1. totalBitsUsed $\leftarrow$ 0
2. for i $\leftarrow$ 1 to sizeof(A)
3. $\{d_i, s_i\} \leftarrow$ Optimization - SingleAttribute(m, k, R_i, n_i) // modify the pramater d and s of A_i
4. totalBitsUsed $\leftarrow$ totalBitsUsed + w(n_i, d_i, s_i, k) // from Equation 3 - 7
5. end loop
6. if totalBitsUsed > m * log2 then
7. totalBitsUsed $\leftarrow$ m * log2
8. end if
9. FpTotal_Before $\leftarrow$ 0
10. for i $\leftarrow$ 1 to sizeof(A)
11. FP[i] $\leftarrow$ $f'(m, k, R_i, n_i, d_i, s_i, \text{totalBits} - w(n_i, d_i, s_i, k))$
12. FpTotal_Before $\leftarrow$ FpTotal_Before + FP[i]
13. end loop
14. iteration $\leftarrow$ 1

// Phase 2 Iteration

15. while TRUE
16. if iteration > 1 then
17. TotalFP_Before $\leftarrow$ TotalFalsePositiveRate(m, k, S, FP) // Function 3 - 1
18. end if
19. For i $\leftarrow$ 1 to sizeof(S)
20. select A_i whose FP[i] is max value in FP, has not been selected yet
21. OptiSingleAttribute(m, k, A_i, S) // Fimction 3 - 2
22. TotalFP_After $\leftarrow$ TotalFalsePositiveRate(m, k, S, FP) // Function 3 - 1
23. end loop
24. if TotalFP_Before = TotalFP_After then // termination condition
25. break
26. end if
27. iteration $\leftarrow$ iteration + 1
28. end loop

Algorithm 3-3 Optimization-MultipleAttributes

Chapter 4 Performance Analysis

In this chapter, the simple experiments on Algorithm 3-1 and Algorithm 3-2 will be simulated, and the evaluation of Division, Overlapping and Division-Overlapping schemes will be presented. The simulation results will be compared with the original method to insert a numerical element into Bloom filter. There are two simulation case of our experiments, one of which is that there was only one numerical attribute inserted into a Bloom filter; the other was that there are more than one numerical attributes inserted into a Bloom filter. In the case of single numerical attribute in a Bloom filter, Algorithm 3-1 was applied to find the optimal parameter (d, s) for single numerical attribute. In the case of multiple numerical attributes in a Bloom filter, we only used Division-Overlapping for our simulation because of its adjustable property that it can transform to pure Division scheme when shift-bit $s = k$ or to pure Overlapping when dividing-range $d = 1$. Algorithm 3-2 was applied to find the optimal parameter (d_i, s_i) for each numerical attribute of the data set.

4.1 Single Numerical Attribute in a Bloom Filter

In single attribute experiments, we first evaluate our schemes in the case of single numerical attribute in a Bloom filter. Algorithm 3-1 will be used to find the optimal parameter (d, s) for minimal false positive rate when inserting many numerical elements into Bloom filter. In Fig. 4-1 There was a numerical attribute inserted into a Bloom filter whose size m was 512, and the number of hash functions k was 8. The number of insertion elements of the query rang in this numerical attribute, whose domain R was 10000, is from 20 to 340. Simulation results of different schemes comparison are presented in Fig. 4-1. When the number of insertion elements was more than 20, the false positive rate of our scheme are better than original. The

results revealed that the Division, Overlapping and Division-Overlapping schemes are better than the original scheme. Fig. 4-1 depicted that the false positive rate of using Overlapping scheme is better than using Division scheme when *n* is less than 260; however, using Division scheme is better than Overlapping scheme when *n* is larger than 260. Result of this figure showed that the Division-Overlapping, which combines Division and Overlapping has minimal false positive rate with its optimal parameter (*d*, *s*).

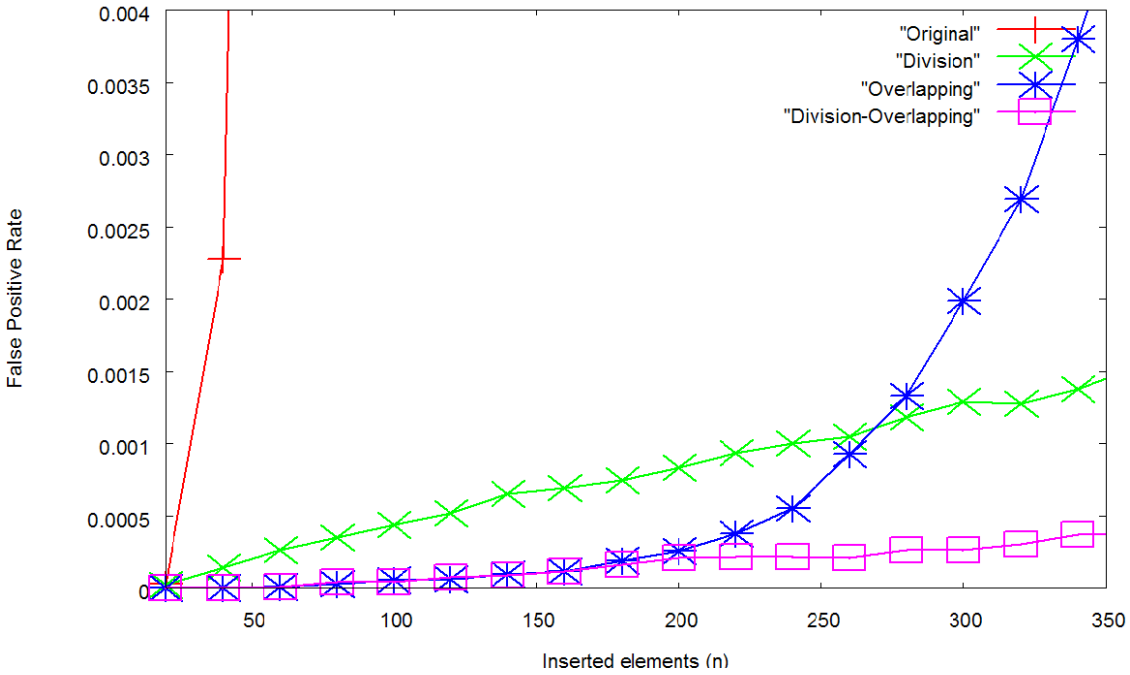


Fig. 4-1 Compare different schemes with optimal configuration

The next part of experiments is to find the relationship between the number of hash functions *k* and the optimal parameters of the numerical attribute. In Fig. 4-2 a numerical attribute, whose domain *R* is 10000, was inserted into an empty Bloom filter. The size of Bloom filter *m* was 512, and the number of hash functions *k* is 8 or 12.The number of insertion numbers *n* of the query range in the numerical attribute was from 100 to 1000. The theoretical value based on Equation 3-13 for false positive rate in Division-Overlapping scheme is approximate to the simulation results. Table 4.1 lists the *k*, (*d*, *s*, bits), where *k* was the number of hash functions used in the

Bloom filter. The (bits) item in the (*d*, *s*, bits) was the number of total insertion bits used in the Bloom filter, and the (*d*, *s*) was the optimal parameter of the Division and Overlapping schemes. The different row of this table means that the Bloom filter used different *k*, and the different column means that the number of insertion elements *n* of the query range was different. The correlations between the number of hash functions *k* and the optimal parameter (*d*, *s*) for false positive rate were slightly different.

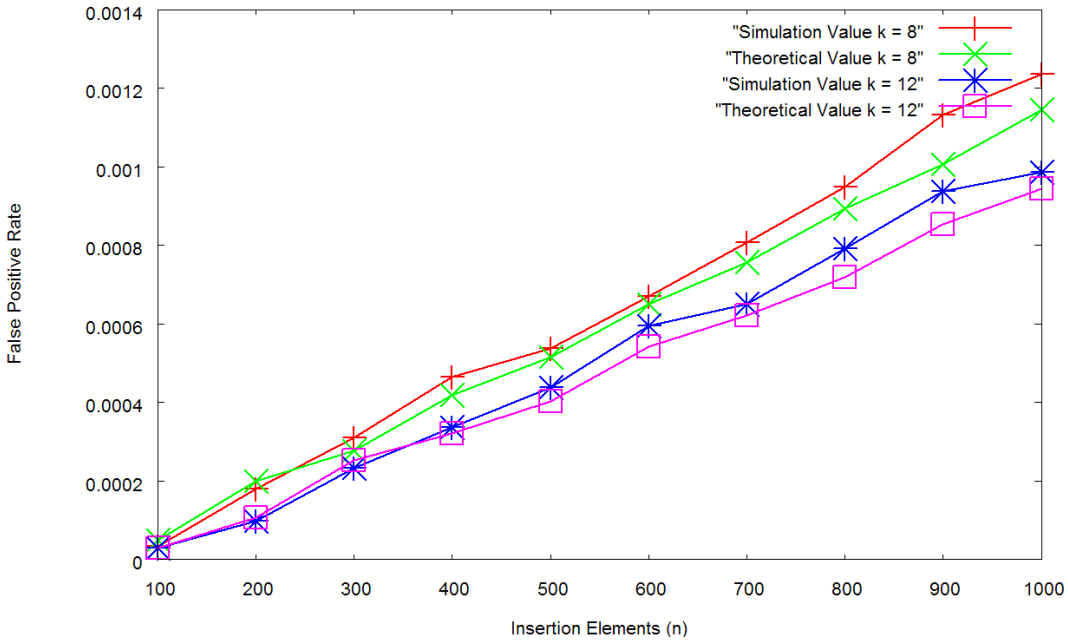


Fig. 4-2 The false positive rate of theory values and simulations results

Table 4-1 The optimal configuration of using different *k*

Insertion Elements (n)	**100**	**200**	**300**	**400**
***k* =8 (*d*, *s*, bits)** **False Positive rate**	(1, 1, 107) 4.85×10^{-5}	(2, 1, 107.5) 1.99×10^{-4}	(2, 1, 157.5) 2.75×10^{-4}	(3, 1, 141) 4.17×10^{-4}
***k* =12 (*d*, *s*, bits)** **False Positive rate**	(1, 2, 210) 2.78×10^{-5}	(1, 1, 211) 1.06×10^{-4}	(1, 1, 311) 2.51×10^{-4}	(2, 1, 211.5) 3.19×10^{-4}
Optimal *k*, (*d*, *s*, bits) **False Positive rate**	14 , (1, 2, 212) 2.65×10^{-5}	13, (1, 1, 212) 1.05×10^{-4}	18, (1, 1, 317) 1.77×10^{-4}	18, (2, 1, 212.5) 3.19×10^{-4}

500	**600**	**700**	**800**	**900**
(3, 1, 174.33) 5.14×10^{-4}	(4, 1, 157.75) 6.5×10^{-4}	(4, 1, 182.75) 7.56×10^{-4}	(5, 1, 167.8) 8.93×10^{-4}	(5, 1, 187.8) 0.001
(2, 1, 261) 4.02×10^{-4}	(3, 1, 211.66) 5.41×10^{-4}	(3, 1, 245) 6.2×10^{-4}	(3, 1, 278.33) 7.17×10^{-4}	(3, 1, 311.66) 8.52×10^{-4}
15, (2, 1, 264.5) 3.91×10^{-4}	17, (2, 1, 316.5) 4.72×10^{-4}	19, (2, 1, 368.5) 5.63×10^{-4}	15, (3, 1, 281.33) 6.97×10^{-4}	16, (3, 1, 315.66) 7.85×10^{-4}

1000
(6, 1, 174.5) 0.0011
(4, 1, 261.75) 9.43×10^{-4}
18, (3, 1, 351) 8.82×10^{-4}

In Fig. 4-2 we find that the false positive rates of using k = 12 is better than using k = 8. In Table 4-1, we also find that using k = 12 used more insertion bits than using k = 8 at every column with different insertion numbers. Back to original Bloom filter optimal relation Equation 2-3, the optimal probability *p* is 1/2, which means that the false positive rate is optimal when the number of insertion bits in Bloom filter was

approximate to m * ln2. In original Bloom filter the optimal false positive is approximate to 0.004= $(1/2)^8$ when m = 512, n = 44 and k = 352, but this may not be applied to our scheme. Because the insertion elements in our case are continuous numbers n within the domain R of the numerical attribute, and the false positive rate function Equation 3-13 is different to Equation 2-2. As a result, the optimal probability p may not be always 1/2 in our Division-Overlapping scheme.

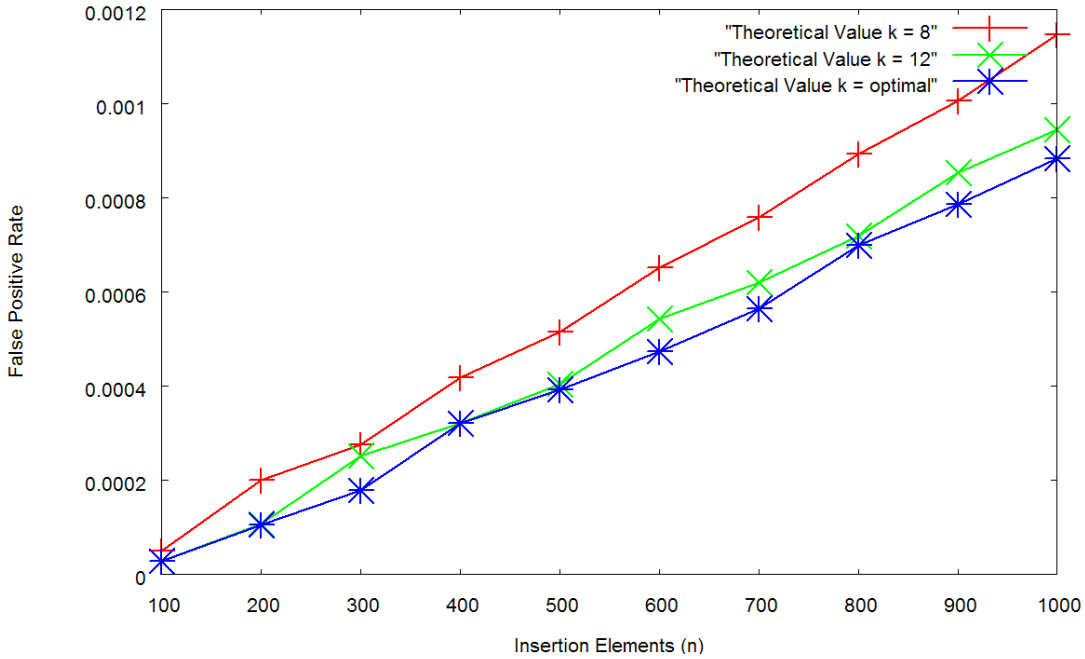


Fig. 4-3 The false positive rate of using different k

Since the optimal number of hash functions k is not constant, the method to find the optimal k is similar to find the inflection point on the false positive rate curve. By Using different k and then finding the optimal parameters d and s of, searching optimal k would stop if the false positive rate of using k+1 with optimal parameters d and s is larger than using k. In Fig. 4-3 we compared the theoretical false positive rate of using optimal k with using k = 8 and using k = 12. The figure showed that using the different k with its optimal parameter (d, s) has optimized false positive rate rather than using constant k = 8 or k = 12. In Table 4-1, the insertion bits of using optimal k are more than using k = 8 and using k = 12 at different the number of insertion

element. Fig. 4-3 depicted that the false positive rate of using optimal k is also better than using k = 8 and using k = 12. The insertion bits of using optimal k in different insertion elements are not always as same as each other because of the additional false positive penalty of Division-Overlapping scheme in Equation 3-8. One possible explanation is that the Division-Overlapping scheme changes the optimal relation between the size of Bloom filter m in original Bloom filter, the insertion elements n and the number of hash functions k.

4.2 Multiple Numerical Attribute in a Bloom Filter

Instead of inserting single numerical attributes into a Bloom filter, we inserted a set of numerical attributes and non-numerical attributes into Bloom filter for our simulation and find the optimal parameter setting (d_i, s_i) for each numerical attribute of the data set according to our proposed optimization Algorithm 3-2. The test data set of multiple attributes in our experiment is the System Defied Attributes (SDA), which used in MFPGC System [13]. Table 4-2 lists the necessary items which were are for querying a user profile, and a SDA might contain one or more non-numerical and numerical attributes. The numerical attributes of the SDA would have many numerical insertion elements of its query range. For example, the numerical attribute “Age” of the SDA, whose domain R is 120 (from 1 to 120) and its insertion elements n is from 1 to 10. The number of insertion bits of each numerical attribute was larger than k when the size of its query range was more than one. Because the number of insertion elements of a numerical attribute was large, and its values of the query range were all continuous numbers, the Division-Overlapping was applied to insert the numerical attributes of the SDA into a Bloom filter. Instead of inserting continuous numbers into Bloom filter, there is only one insertion element of non-numerical attributes because the value of non-numerical attribute contained only string-type value.

Table 4-2 System Defined Attribute

Attribute Name	Attribute Type	Attribute Value
Name	String (non-numerical)	Random String in length 20
Nick Name	String (non-numerical)	Random String in length 20
University	String (non-numerical)	Random String in length 20
Hobby	String (non-numerical)	Random String in length 20
Professional	String (non-numerical)	Random String in length 20
Age	Integer [1:120] (numerical)	Query Range: 1~10
Year	Integer[1900:2100] (numerical)	Query Range: 2~20
Income	Integer[0:5000000] (numerical)	Query Range: 50000~5000000
Longitude	Integer[-1800000: 1800000] (numerical)	Query Range: 10~100
Latitude	Integer[-900000: 900000] (numerical)	Query Range: 10~100

In our multiple attribute experiments, there were five non-numerical attributes and five numerical attributes in the SDA. The five numerical attributes were inserted into Bloom filter by our numerical attribute representation scheme Division-Overlapping. To decide the optimal dividing-range *d* and shift-bit *s* for each numerical attributes, our optimization Algorithm 3-2 was used to find the optimal parameters for each numerical attribute. In Fig. 4-4, we inserted the data set of multiple attributes listed in Table 4-2 into a Bloom filter, whose size *m* is 512 and the number of hash function *k* is 8 or 12. The simulation results depicted in Fig. 4-4 showed that the false positive rates of simulation results are consistent with the theoretical values based on Equation 3-15. Clearly, the false positive rate of using *k* = 12 with the optimal *d* and *s* for each numerical attribute of the SDA is better than using k = 8. We found that the number of insertion bits is a key factor to affect the false positive rate in our Division-Overlapping scheme. Like the case of single

numerical attribute in a Bloom filter, the false positive rate of using more insertion bits is better. Moreover, in Fig. 4-5 we compare the theoretical false positive rates of using optimal *k=15* in multiple attributes with using k = 8 and k = 12, using optimal k=15 is better than the others.

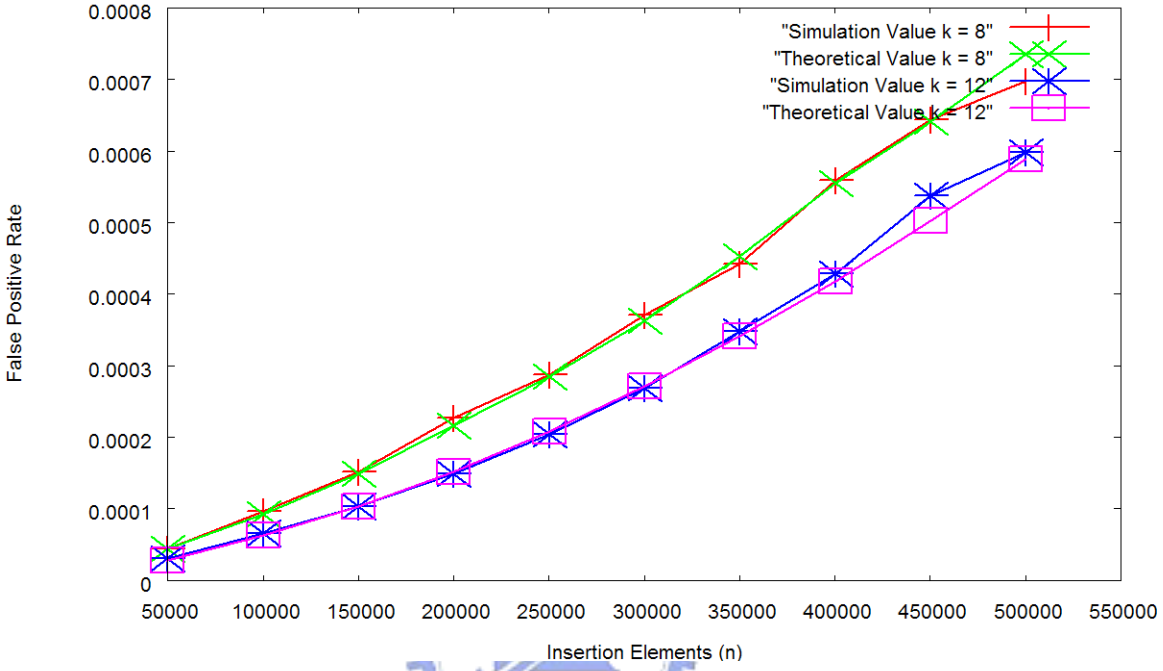


Fig. 4-4 The false positive rate of theory values and simulations results

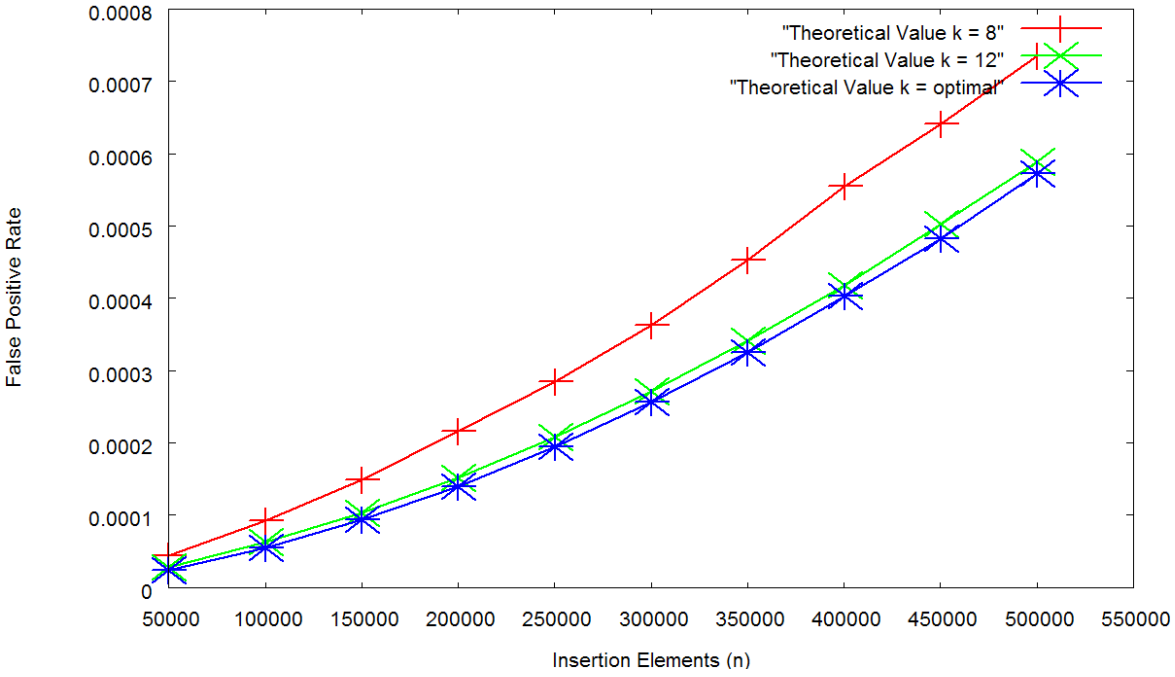


Fig. 4-5 The false positive rate of different k

Table 4-3 and 4-4 summarized the optimal parameter (d_i, s_i) of each numerical attribute of the SDA and their false positive rates when inserting 500235 elements into a Bloom filter. In Table 4-3, the number of hash function k is 8, and the number of insertion bits of each attribute is different to each other. If the attribute was non-numerical like “Name”, “Nick Name”, “University”, “Hobby” and “Profession”, the number of insertion bits is k because the random string in different length can be hashed to only k random values by k hash functions; however, if the attribute was numerical like “Age”, “Year”, “Income”, “Longitude” and “Latitude”, the number of insertion bits is decided by the parameter (d, s) of the Division-Overlapping scheme. In Table 4-3 and 4-4, the number of insertion bits of “Income” attribute is larger than the other attributes, and its false positive rate is also larger than the false positive rate of the others. This is the effect that there were 500000 insertion element of “Income” attribute; as a result, the dividing-range d was so large to compress the continuous numbers. The penalty of dividing range error would be too large, so the false positive rate of “Income” attributes became the main factor to affect the total false positive rate of all attributes in the SDA. According to parameter optimization Algorithm 3-2, the optimal parameter of each numerical attribute would be determined for the optimal average false positive rates in the test data set.

Results of the optimal parameter of using the optimal k = 15 are presented in Table 4-4. The number of total insertion bits of using k = 15 is larger than using k = 8, and the average false positive rate of using k =15 is smaller than using k = 8. The insertion bits percentage of each attribute in Table 4-4 is similar to Table 4-3, but the false positive rate of each attributes which used k =15 is smaller than using k = 8 in Table 4-3. In Table 4-3 and 4-4, we can find that the optimal parameter d and s of each numerical attributes under different k would be different. Like the result of single attribute in a Bloom filter, using more insertion bit in the data set of multiple

attributes had better correctness. The number of total insertion bits is associated with the number of hash functions *k*. Obviously, the optimal average false positive rate of multiple attributes is correlated with not only the number of hash functions *k* but also the optimal parameter dividing-range *d* and shift-bit *s* of Division-Overlapping scheme for each numerical attribute when there were numerical attributes in the data set.

Table 4-3 The optimal parameter of using $k = 8$

Attribute Name	**Age**	**Year**	**Income**	**Longitude**
Insertion Elements (*n*)	10 numbers	20 numbers	500000 numbers	100 numbers
Individual (d, s, bits)	(1, 3, 35)	(1, 2, 43)	(9315, 1, 62.78)	(100, 1, 8.99)
Individual FP rate	6.51×10^{-4}	0.00131	0.00399	9.63×10^{-5}

Latitude	**Name**	**Nick Name**	**University**	**Hobby**
100 numbers	Random string	Random string	Random string	Random string
(81, 1, 9.22)	(-, -, 8)	(-, -, 8)	(-, -, 8)	(-, -, 8)
1.2×10^{-4}	1.27×10^{-4}	1.27×10^{-4}	1.27×10^{-4}	1.27×10^{-4}

Profession	
Random string	**Total Insertion Element: 500235**
(-, -, 8)	**Total Insertion Bits: 201.94**
1.27×10^{-4}	**Average False Positive: 7.35×10^{-4}**

Table 4-4 The optimal parameter of using optimal $k = 15$

Attribute Name	**Age**	**Year**	**Income**	**Longitude**
Insertion Elements (*n*)	10 numbers	20 numbers	500000 numbers	100 numbers
Individual (d, s, bits)	(1, 6, 69)	(1, 4, 91)	(5129, 1, 112.48)	(65, 1, 16.52)
Individual FP rate	3.88×10^{-4}	9.08×10^{-4}	0.00365	8.14×10^{-5}

Latitude	**Name**	**Nick Name**	**University**	**Hobby**
100 numbers	Random string	Random string	Random string	Random string
(46, 1, 17.15)	(-, -, 15)	(-, -, 15)	(-, -, 15)	(-, -, 15)
9.73×10^{-5}	6.34×10^{-5}	6.34×10^{-5}	6.34×10^{-5}	6.34×10^{-5}

Profession	
Random string	**Total Insertion Elements: 500235**
(-, -, 15)	**Total Insertion Bits 381.15**
6.34×10^{-5}	**Average False Positive: 5.72×10^{-4}**

4.3 The Discrepancy of Analytic and Simulation

The performance of our schemes depends on the hash function randomization. The hash function class used in this thesis is MD5, and the k random indices of an insertion element are derived from 128-bit hash value. We assume that the hash function is perfect random and different elements whose hash values would be always different to each other. The previous analytic value of false positive rate equation is based on the assumption that the hash function is perfect random. However, the simulation results would be different to the analytic value if we used a specific numerical attribute name and numerical range R. In Fig. 4-6, we inserted a numerical

attribute named “Age” by Division-Overlapping scheme with its optimal parameter the query range was from 4 to 60 where the domain *R* of the attribute is 120. The emulation results of this figure is the statistic false positive rate where we considered all possible false positive occurrence cases rather than randomly chose the query range and test numerical element. Result of Fig. 4-6 showed that the simulation values are approximate to emulation values, but it is extremely different to the analytic values when the query range was larger than 44. As we have mentioned before in chapter 2, the elements of a numerical attribute were concatenated with the attribute name and then inserted into Bloom filter. The random indices of inserted elements with same attribute name may have overlaps when the query range getting large if the hash value of each element is similar to its neighbor elements. Therefore, the false positive rate is also correlated with the randomization performance of hash function. The discrepancy of analytic values and simulation would occur when the number of inserted elements of a specific numerical attribute was larger than about 1/3 domain.

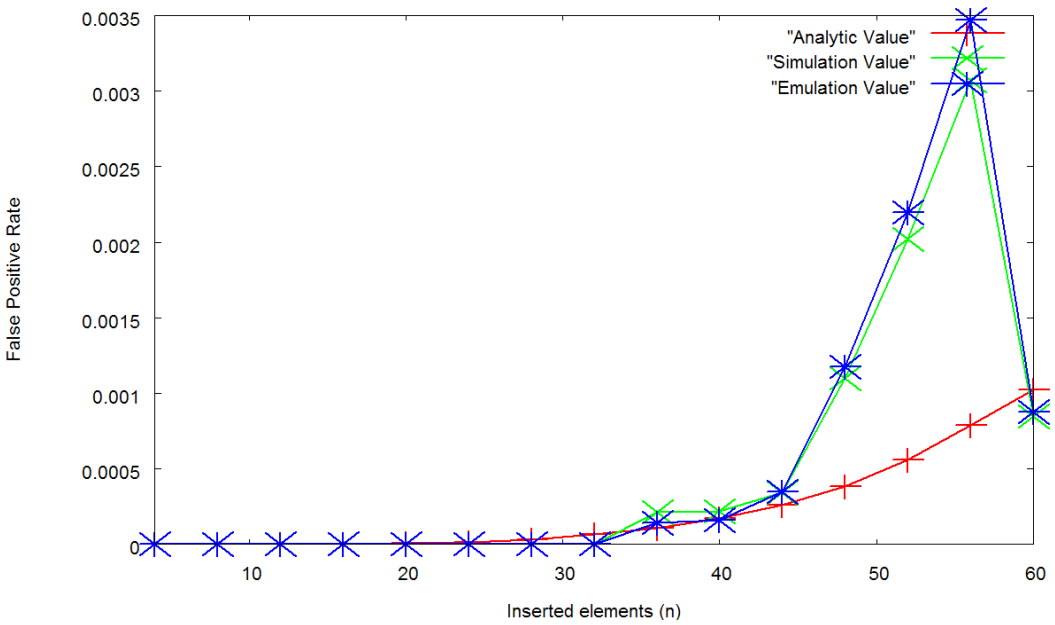


Fig. 4-6 The discrepancy of Analytic and Simulation

Chapter 5 Conclusions

In this thesis we present new Bloom filter design for numerical range to reduce the false positive rate even when a large range of numerical elements is inserted. When using the traditional Bloom filter design, the false positive rate increases exponentially as the number of inserted elements increases. The Division and Overlapping scheme first reduces the number of insertion bits by overlapping the insertion bits of consecutive numbers, i.e., $1< o < k$ and $d=1$. If the number of insertion bits is still too large for the Bloom filter, the Division and Overlapping scheme group consecutive numbers into divisions to reduce the number of elements inserted, i.e., $o = k-1$ and $d>1$. Using the Division and Overlapping scheme, the false positive rate only increases linearly as the number of inserted elements increases. We show that the optimal configuration of Bloom filter representing a numeric range of single attribute can be obtained, i.e., the false positive rate is minimized. In addition, we developed a heuristic algorithm to obtain near optimal configurations for multiple attributes.

This thesis has taken a step in the direction of reducing the false positive and finding the optimal relation between the number of hash functions and the parameters of our numerical compression schemes. However, the optimal configurations of Bloom filter representing multiple string/numerical-range attributes remains an open problem. For the optimal configuration, the number of inserted bits for each attributes can be different. The goal is to minimize the total false positive rate. More investigation is needed on the optimal configurations. In addition, it is important to consider the domain of a numerical attribute.

Reference

[1] Ion Stoica_, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan “Chord: A scalable peer-to-peer lookup service for internet applications” Proceedings of the 2001 SIGCOMM conference, 2001

[2] Sylvia Ratnasamy , Paul Francis , Mark Handley , Richard Karp , Scott Schenker, “A scalable content-addressable network”, Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, p.161-172, August 2001

[3] F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. “PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities.” Technical Report DCS-TR-487, Rutgers University, Sept. 2002.

[4] Patrick Reynolds and Amin Vahdat “Efficient Peer-to-Peer Keyword Searching”, Proceedings of International Middleware Conference, 2003

[5] L Fan, P Cao, J Almeida, AZ Broder “Summary cache a scalable wide-area web cache sharing protocol”, IEEE/ACM Transactions on Networking (TON), 2000

[6] M Mitzenmacher “Compressed bloom filters”, IEEE/ACM Transactions on Networking (TON), 2002

[7] A Kirsch, M Mitzenmacher “Distance-Sensitive Bloom filter”, Proceedings of the Eighth Workshop on Algorithm Engineering 2006

[8] S Cohen, Y Matias “Spectral Bloom Filter”, Proceedings of the 2003 ACM SIGMOD international conference

[9] A Kumar, J Xu, J Wang “Space-code bloom filter for efficient traffic flow measurement”, Selected Areas in Communications, IEEE Journal on, 2006

[10] D Guo, J Wu, H Chen, X Luo “Theory and Network Application of Dynamic Bloom Filters”, Proceedings of the 25th Annual Joint Conference of the IEEE 2006

[11] K Xie, Y Min, D Zhang, J Wen, G Xie “A Scalable Bloom Filter for Membership Queries”, Global Telecommunications Conference, 2007

[12] BH Bloom “Space/Time Trade-offs in hash Coding with Allowable Errors”, Communications of the ACM, 1970

[13] Hou-Wei Lee “Multiple-Function Personal/Group Communication System” NCTU, 2007