行政院國家科學委員會專題研究計畫 成果報告

多功能個人/群體通訊系統設計(II)
研究成果報告(精簡版)

計 畫 類 別 ：個別型
計 畫 編 號 ：NSC 97-2221-E-009-059-
執 行 期 間 ：97 年 08 月 01 日至 98 年 07 月 31 日
執 行 單 位 ：國立交通大學資訊工程學系（所）

計 畫 主 持 人 ：張明峰

處 理 方 式 ：本計畫可公開查詢

中 華 民 國 98 年 10 月 26 日

行政院國家科學委員會專題研究計畫成果報告

多功能個人/群體通訊系統設計

Multi-Function Personal/Group Communication System Design (II)

計畫編號：97-2221-E-009-059-
執行期限：2008.08.01 至 2009.07.31

主持人：張明峰　交通大學資工系
計畫參與人員：蔡玄亞、廖威凱、游伯瑞、李茂弘　交通大學資工系

中文摘要

現今的通訊系統，包含電信系統、即時通訊、電子郵件、和網路電話等，都利用明確的使用者識別碼來指定使用者。如不知道接收端的識別碼即不能通訊。本計畫第一年開發一多功能通訊平台（MFPGC）能利用指定接收端的各項屬性，例如姓名、年紀，學校等，建立通訊連線。MFPGC 架構在 Chord 之上，使用布隆過濾器(Bloom filter)來儲存多屬性的資料，包含字串，數值和混合型態的屬性。透過屬性發佈和搜尋的機制，媒合發話端和受話端。本計畫第二年我們針對 MFPGC 中布隆過濾器，提出創新的設計，使其能有效率的表示數值範圍的屬性，例如指定接收端需位於東經 121.1000°-121.1999°，北緯 25.0000°-25.0999°。傳統的布隆過濾器無法表示如此大的數值範圍。我們針對數值範圍內的元素，加以群組化，或/且重疊化。群組化的方式減少加入布隆過濾器的元素，而重疊化的方式減少布隆過濾器中設定為 1 的位元數。為了降低所有屬性之總體錯誤肯定率（false positive rate），我們開發了一兩階段的程式尋找群組與重疊化布隆過濾器的最佳參數設定。第一階段利用經濟學的邊際效益理論取得初步參數設定；第二階段利用梯度下降(gradient descent)演算法求得更好的參數設定。我們使用數值分析的方法驗證群組與重疊化布隆過濾器，以及參數最佳化程式的效益。我們的實驗結果顯示對較小範圍的數值，應只採用重疊化技巧；當數值範圍大過臨界點時，則應採用群組化和重疊化兩種技巧。而且，群組與重疊化布隆過濾器確實能表示大範圍的數值，並提供很低的錯誤肯定率，這是傳統的布隆過濾器所無法做到的。

關鍵字：網路通訊，多屬性查詢，數值範圍查詢，布隆過濾器

Abstract

Communication services today, such as telephony, instant message, email, and VoIP, use a specific user or device ID to specify the called party. If the callee's ID is unknown, communication becomes impossible. Another way to indicate the callee(s) is to specify the callee's attributes, such as the callee's name, age, etc. A set of user attributes, which is meaningful and easy to remember, can be used to set up a communication. Developed in the first year of this project, a Multi-Function Personal/Group Communication (MFPGC) system supports communications using specific IDs and/or multiple under-specified attributes. Communications using multiple user attributes is feasible through publishing and querying users' attributes on a DHT. The DHT of MFPGC system is based on Chord, and Bloom filter is used to represent user attributes, which can be string, numeric, and hybrid data. In the second year of this project, we designed a new Bloom filter that can represent large ranges of numeric data in a space-efficient way. The techniques we used are grouping and/or overlapping the elements of the inserted numeric ranges. The design also supports data/range queries of multiple attributes. The grouping scheme reduces the number of keys inserted into a Bloom filter, while the overlapping scheme reduces the number of bits set to one in a Bloom filter. In addition, these two schemes can be combined to provide both benefits. To minimize the total false positive rate of all attributes, we have developed a two-phase algorithm using the marginal utility theory and gradient descent to optimize parameter configurations of a combined grouping and overlapping Bloom filter. Numeric analysis has also been conducted to demonstrate the effectiveness of our modified Bloom filter and the proposed configuration algorithm. The numeric results show that as the size of an inserted numeric range increases, overlapping scheme should be used first, and then both overlapping and grouping need to be used. The combined grouping and overlapping Bloom filter significantly reduces the false positive rates of data sets with large ranges of numeric data.

Keywords: Bloom filter, numeric range query, multiple-attribute query

I. Introduction

Bloom filter was invented by B. H. Bloom to represent a data set and support membership queries [1]. It is a space-efficient randomized data structure with a probability of false positives. In other words, elements that are not in the data set may be falsely determined to be in. In recent years, Bloom filter has been used in various network applications where the saving of space and/or transmission bandwidth is important [2]. Squid, a distributed web cache system, uses Bloom filter to represent the summary of files cached [3]. OceanStore, a distributed file storage, uses a modified Bloom filter to indicate the files stored in the neighbors of each node [4]. Bloom filter was used to send the intermediate results of a multi-keyword search to save transmission bandwidth required [5]. Bloom filter has also been used in packet routing [6-8] and traffic measurement [9].

Bloom filter has a number of extensions. One major limitation of Bloom filter is that deleting an element from a traditional Bloom filter may not be possible because of the way it is constructed. Counting Bloom filter was designed to solve this problem [10]. Each entry in a counting Bloom filter is exactly a counter. Inserting an element is to hash the element and increment the corresponding counters. On the other hand, deleting an element is to hash and decrement the corresponding counters. To be more space-efficient, Mitzenmacher presented compressed Bloom filter that has the same false positive rate as a traditional Bloom filter but with fewer bits [11]. A larger and sparser Bloom filter was first constructed, and then compressed. Guo et al. presented multi-dimension dynamic bloom filters (MDDBF) that dynamically adjust the number of Bloom filters used to represent a data set according to the size of the data set [12].

However, Bloom filter has not been used to represent a large range of numeric data because of space inefficiency. The proper size of a Bloom filter can be determined by a desired false positive rate and the number of inserted elements. Given a false positive rate *f*, a

Bloom filter needs $1.44 \times \log_2(1/f)$ bits of space per inserted element [2]. As a result, a Bloom filter is space-inefficient in representing a large range of numeric data in a straightforward manner. For example, a square area of Latitude 25.0000°-25.0999° and Longitude 121.1000°-121.1999° represents approximately 1 km by 1 km square area. To be accurate to the four decimal places, the area can be represented by two integer intervals [250000, ..., 250999] and [1211000, …, 1211999]; each range consists of 1000 integers. Note that the precession of this representation is about 10 meters. This data type is valuable in supporting location-based communications, for example, broadcasting messages to users in this area. However, to obtain a false positive rate of 0.001, a traditional Bloom filter needs approximately 48K bits, while the size of a typical Bloom filter is less than 1K bits. It is obvious that a traditional Bloom filter is unsuitable for representing large ranges of numeric data. A new space-efficient design is essential to apply a Bloom filter to versatile applications.

The objective of this project is to develop a new Bloom filter to represent multiple data attributes, which may include large ranges of numeric data. The new design should be space-efficient to provide a small probability of false positives for membership queries, as a traditional Bloom filter does for data sets of atomic data types. Our new Bloom filter design can be easily applied to various applications. In particular, it can be used in multiple attribute range queries [13, 14] where the use of Bloom filter has never been considered.

The remainder of the report is organized as follows. Section II describes our new Bloom filter designs for representing numeric ranges. Analytic models to obtain the false positive rates of the new designs are presented in Section III. Section IV describes a two-phase optimization algorithm using marginal utility theory and gradient descent to obtain a near optimal configuration of the modified Bloom filter. Section V discusses the simulation and analytic results, while Section VI concludes the report.

II. Bloom Filter Design for Numeric Ranges

A traditional Bloom filter is an array of m bits representing a data set of n elements $\{x_1, x_2, \ldots, x_n\}$. Let S denote a data set. All bits of the Bloom filter are 0 initially. k independent hash functions $h_1, h_2, \ldots, h_k$, with range $\{1, \ldots, m\}$, are used to hash each element x in S to k array positions $h_1(x), h_2(x), \ldots, h_k(x)$, and the bits at the hashed array positions are set to 1. To check membership of y in S, we verify if all bits $h_i(y)$, i=1, 2, .., k, are set to 1. If at least one bit is 0, we are sure that y is not in S; otherwise, y is assumed to be in S, with a probability of being wrong. The situation of being wrong is so called *false positive*. Figure 1 depicts an example of Bloom filter representing $\{x_1, x_2\}$. From the Bloom filter, we can determine that y_1 is not in the set because not all the hashed array positions are set to 1. On the other hand, although y_2 is not in the set, the Bloom filter cannot distinguish this because all the hashed array positions are all set to 1. Thus y_2 is a false positive.

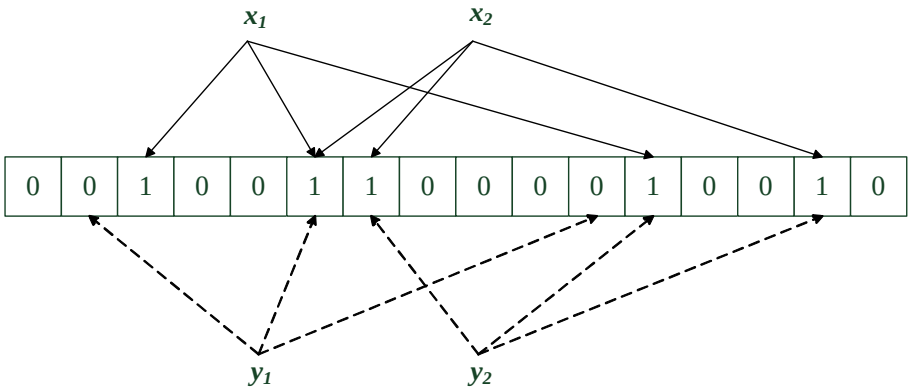

Figure 1. An illustrative example of Bloom filter operation.

Assume that the hash functions are perfectly random, the probability of a false positive for an element is not in S (i.e., the false positive rate) can be derived as follows. Given m, n, and k of a Bloom filter, the probability that an array position is not set to 1 after n elements are hashed into the Bloom filter can be approximated as

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

Let *f* denote the false positive rate. An element that is not in the set would be determined as a false positive if all its *k* hashed array positions are set to 1. Thus *f* can be expressed as follows.

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

The false positive rate is minimized when $k = (m/n)\ln 2$ and $p = 1/2$, i.e., when half of the array positions are set to 1 [2].

In this project, we intend to use a Bloom filter for multi-attribute data/range queries. A data set, *S*, contains multiple data attributes. Each attribute is specified by an attribute name and an attribute value. The attribute values can be atomic or numeric ranges. An atomic data value can be a text string or an integer. Floating point numbers can be represented by integers with a desired degree of accuracy. For example, Latitude 23.2621448° can be represented by 232621 with accuracy to the four decimal places, i.e, 10 meters. In addition, the attribute value can be a numeric range, for example, Latitude 23.0000°-23.999°. Hashing an attribute into a Bloom filter is to hash the text string concatenating the attribute name and the attribute value. For example, inserting Latitude 23.0000° is to set array positions h_i ("Latitude:230000") to 1. For multiple-attribute queries, we are interested in checking if a data set S_1 is a subset of another set S_2. Let $BF(S)$ denote the Bloom filter representing data set *S*. It is clear that if all array positions that are set to 1 in $BF(S_1)$ are also set in $BF(S_2)$, we can assume S_1 is a subset of S_2 with a possibility of being wrong.

We will describe three approaches to modify a traditional Bloom filter for multi-attribute queries. We focus on how a numeric range is represented by the modified Bloom filter; attributes of atomic data type are treated in the same way as the traditional Bloom filter. The techniques we used are grouping successive elements (i.e., successive numbers of a numeric

range) and overlapping hashed positions of successive elements.

1. Grouping successive elements

The problem of representing a large range of numeric data by a traditional Bloom filter is simply that there may be too many elements inserted. When a large number of elements are inserted into a Bloom filter and most of the bits in the Bloom filter are set to 1, false positives increase, and thus the Bloom filter becomes useless. A straightforward solution for the problem is to reduce the number of elements inserted. This can be done by equally dividing the numeric scope into groups; each group consists of a fixed number of successive numbers (elements). The numbers in the same group are represented by the same inserted key, i.e. their hashed positions in a Bloom filter are the same. This approach will be referred to as *grouping Bloom filter*. Note that this approach reduces the number of bits set to 1 in a Bloom filter representing a large range of numeric data, because fewer keys are inserted. Figure 2 depicts the hashed positions of an example grouping Bloom filter, where five successive numbers are grouped together and five hashed array positions are generated for each group. Ages 0-4 are all represented by Age 0 when inserted in the Bloom filter. In other words, the grouping Bloom filter does not distinguish between Ages 0-4.

Since numbers in the same group are represented by the same key, the number of keys inserted and the number of bits set to 1 in the Bloom filter are reduced at the cost of false positives caused by numbers in the same group. If some numbers of a group are inserted in a

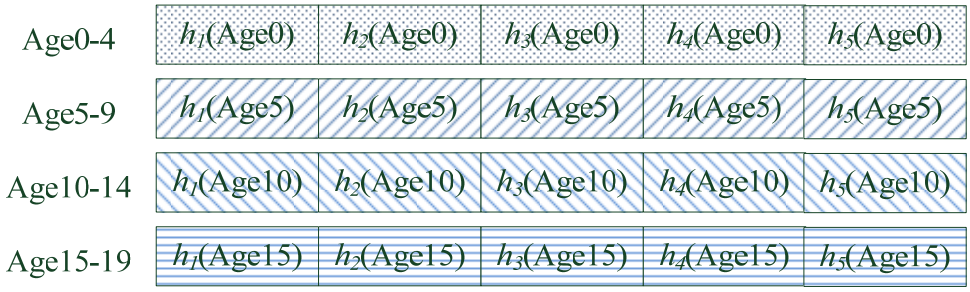

Figure 2. An example of hashed array positions for a grouping Bloom filter.

Bloom filter, other numbers of the group would be determined as false positives by the Bloom filter. The false positives caused by grouping successive numbers will be referred to as *grouping false positives*.

2. Overlapping hashed positions of successive elements

Another solution for inserting a large range of numbers in a Bloom filter is to reduce the number of array positions set to 1 by overlapping the hashed positions of successive numbers. In a traditional Bloom filter, k independent hash functions hash each inserted element to k array positions, and the positions are set to 1. In this overlapping scheme, each inserted number also sets k array positions to 1, but only some, s ($s < k$), of the k positions are determined by hashing the inserted number. The remaining (k-s) of the hashed positions are determined by hashing the successive numbers, i.e., (k-s) of the k hashed positions of an inserted number are the same as those of its successive number. Figure 3 depicts hashed positions of an example overlapping Bloom filter, where $k = 5$ and $s = 2$. The array positions set by Age 0 are h_1(Age0), h_2(Age0), h_1(Age1), h_2(Age1), and h_1(Age2). Only two array positions are determined by Age 0; the other three array positions are determined by Age 1 and Age 2. By overlapping the inserted bits of successive numbers, the number of array positions set to 1 by a numeric range is reduced. The cost of this approach is that numbers adjacent to an inserted range have higher false positive rates, because their hashed array positions overlap with those of the inserted numbers. This will be referred to as *overlapping*

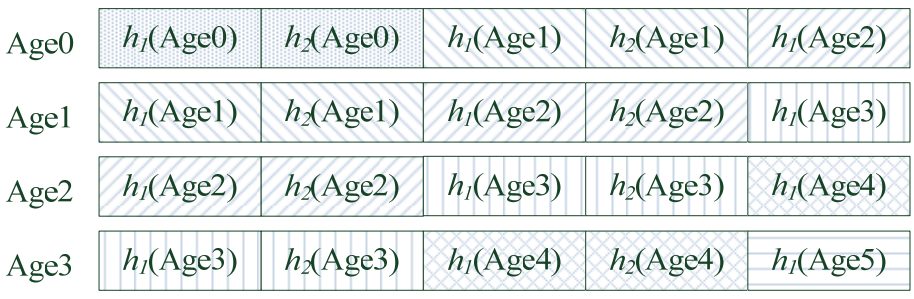

Figure 3. An example of hashed array positions for an overlapping Bloom filter.

false positives.

3. Grouping-overlapping Bloom filter

To further reduce the number of inserted bits in a Bloom filter, we combine both the grouping and overlapping techniques. For grouping-overlapping technique, a numeric scope is equally partitioned into several groups and each group consists of successive numbers. Each group uses *s* independent hash functions to generate *s* array positions. When a number of a specific group is inserted, in addition to the *s* hashed positions, (*k*-*s*) array positions generated by the successive groups are set to 1. Figure 4 depicts the hashed positions of an example grouping and overlapping Bloom filter. Five successive numbers (e.g., Ages 0-4) form a group. Each group generates two hashed positions and use additional three hashed positions of the successive groups. It is clear that a grouping-overlapping Bloom filter has both grouping false positives and overlapping false positives as we have described.

III. Analytic Models

In this section, we theoretically analyze our designed Bloom filters. Indeed, we first estimate the number of array positions that are set to 1, and further derive the probabilities of grouping/overlapping false positives.

Let *R* and *n* denote the scope size of a numeric attribute and the size of an inserted numeric range, respectively. Assume that the size of a Bloom filter is *m* bits, and the number

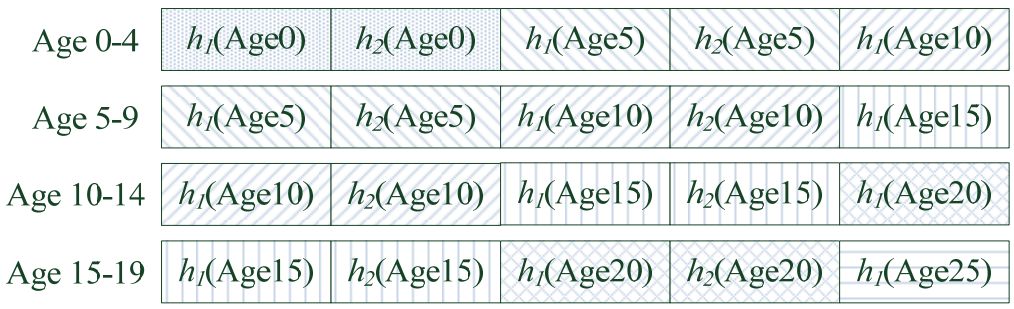

Figure 4. An example of hashed array positions for a grouping-overlapping Bloom filter.

of array positions set by each inserted key is *k*. The false positive rate of each approach can be derived as follows.

1. Grouping Bloom filter

We first consider a grouping Bloom filter. Let *d* denote the size of a group, i.e., the number of discrete numeric values per group. Let *g*(*n*,*d*,*k*) denote the expected number of hashed positions generated by a numeric range of size *n*; note that the hashed array positions may be duplicated. *g*(*n*,*d*,*k*) is derived through Eq. (3).

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

After setting *g*(*n*,*d*,*k*) hashed positions to 1, the probability that a bit position in the Bloom filter is still 0 can be expressed as follows.

$$p_g = (1-\frac{1}{m})^{g(n,d,k)} \approx e^{-(\frac{n-1}{d}+1)k/m} \tag{4}$$

The elements that may cause grouping false positives are those that are not in the inserted range, but in the same group with the inserted keys. The expected number of those elements can be expressed as in Eq. (5).

$$\frac{\sum_{i=0}^{d-1} i + (n+i) \bmod d}{d} = \frac{d(d-1)}{d} = d-1 \tag{5}$$

Let f_g denote the false positive rate of a grouping Bloom filter; f_g can be expressed as Eq. (6). The first term to the right of the equal sign in Eq. (6) represents the probability of grouping false positives, and the second term represents the probability of false positives that are not caused by grouping.

$$f_g = \frac{d-1}{R-n} + \frac{R-n-(d-1)}{R-n}(1-p_g)^k \tag{6}$$

2. Overlapping Bloom filter

Let *s* denote the number of hashed positions generated by each inserted element. When

inserting a numeric range of size *n*, the number of hashed positions generated can be expressed as follows,

$$o(n,s,k) = (n-1)s + k \tag{7}$$

After setting *o*(*n*,*s*,*k*) hashed positions to 1, the probability that a bit position in the Bloom filter is still 0 can be expressed as follows,

$$p_o = \left(1 - \frac{1}{m}\right)^{o(n,d,k)} \approx e^{-\left((n-1)s+k\right)/m} \tag{8}$$

The number of elements that may cause overlapping false positives at each end of the inserted numeric range can be expressed as in Eq. (9).

$$r = \lceil k/s \rceil - 1 \tag{9}$$

The false positive probability of an overlapping Bloom filter can be expressed as in Eq. (10). The first term to the right of the equal sign in Eq. (10) represents the probability of overlapping false positives, and the second term represents the probability of false positives that are not caused by overlapping.

$$f_o = \frac{2}{R-n} \sum_{i=1}^{i \le r} (1 - p_o)^{is} + \frac{R-n-2r}{R\text{-}n} (1 - p_o)^k \tag{10}$$

3. Grouping-overlapping Bloom filter

Let *d* denote the size of a group, and *s* denote the number of generated hashed positions per group. Note that total number of hashed positions of each group is *k*, but (*k*-*s*) of which are generated by successive groups. When inserting a numeric range of size *n*, the expected number of hashed positions generated can be expressed as follows,

$$go(n,d,s,k) = \left(\sum_{i=0}^{d-1} \lceil (n+i)/d \rceil / d - 1\right)s + k = \frac{n-1}{d}\mathrm{s} + k \tag{11}$$

After setting *go*(*n*,*d*,*s*,*k*) hashed positions to 1, the probability that a bit position in the Bloom filter is still zero can be expressed as follows,

$$p_{go} = (1 - \frac{1}{m})^{go(n,d,s,k)} \approx e^{-(\frac{n-1}{d}s+k)/m} \quad (12)$$

The number of groups that may cause overlapping false positives at each end of the inserted numeric range can be expressed as,

$$r = \lceil k/s \rceil - 1. \quad (13)$$

The false positive probability of an overlapping Bloom filter can be expressed as Eq. (14). The first term to the right of the equal sign in Eq. (14) represents the probability of grouping false positives, the second term the probability of overlapping false positives, and the last term represents the probability of false positives that are caused by neither grouping nor overlapping.

$$f_{go} = \frac{d-1}{R-n} + \frac{2d}{R-n}\sum_{i=1}^{r}(1-p_{go})^{is} + \frac{R-n-(d-1)-2rd}{R-n}(1-p_{go})^{k} \quad (14)$$

IV. Near Optimal Configurations of the Proposed Bloom Filters

For grouping and overlapping Bloom filters, the optimal choices of *k*, *d* and *s* are the key factors to minimize the false positive rates. First, we consider a simple situation where only a single numeric range is inserted in a Bloom filter. Although this simple situation may seem impractical, it can clearly demonstrate how the false positive rate of each scheme holds as the range of the inserted numeric data increases. Followed, we will consider the optimal configurations of a Bloom filter representing a multi-attribute data set with numeric ranges.

1. The configuration for a single numeric range

To fairly compare the performance of different schemes, the optimal configuration, i.e., the best choice of *k*, *d* and *s*, for a given numeric range should be used. Note that the grouping-overlapping Bloom filter is the most general of all schemes; other schemes are the special cases of the grouping-overlapping Bloom filter. Indeed, the traditional Bloom filter is

a special case where $d = 1$ and $s = k$, the grouping Bloom filter $d \geq 1$ and $s = k$, and the overlapping Bloom filter $d = 1$ and $1 \leq s \leq k$. Since the grouping-overlapping Bloom filter is the most general design, Therefore, we only describe an algorithm that optimizes the configuration of a grouping-overlapping Bloom filter. The optimal configurations of other schemes can be obtained in similar ways.

Given the scope size of a numeric attribute (R) and the size of the inserted range (n), we need to find the optimum choice of (k, d, s) for a grouping-overlapping Bloom filter of size m bits to minimize the false positive rate. From the experience of a traditional Bloom filter, we conjecture that the optimal configuration sets approximately half of the array positions (i.e., $m/2$) by a total number of $m\ln 2$ hashed positions. Later experiments support this conjecture. From Eq. (11), we have

$$\frac{n-1}{d} s + k = m \times ln2 \tag{15}$$

When n is large, s/d (named s-to-d ratio) can be approximated as in (16).

$$\frac{s}{d} \approx \frac{m\ln 2}{n} \tag{16}$$

Note that s-to-d ratio represents the approximated number of hashed positions for each inserted element. From our initial experiments, we observed that the optimal choice of (d, s) is in the form of either (1, s) or (d, 1). For example, (2, 7) performs poorer than (1, 3) or (1, 4). The reason is grouping produces more false positives than overlapping. This will become clear later when we compare the numeric results of all schemes. Therefore, the initial setting of (d, s) is determined as in (17).

$$(d, s) = \begin{cases} (1, \text{round}(\frac{s}{d})), & \text{if} \quad \frac{s}{d} \geq 0.7 \\ (\text{round}(\frac{d}{s}), 1), & \text{otherwise} \end{cases} \tag{17}$$

Given (d, s), the optimal k that minimizes the false positive rate can be easily obtained

because the false positive rate appears to be a concave function of k. On the other hand, arranging the possible choices of (d, s) pair in a sequence with an increasing order of s/d, we obtain a sequence as follows, (…, (3, 1), (2, 1), (1, 1), (1,2), (1,3), …). Note that the number of hashed positions of an inserted range increases as s/d increases. Our experiments show that the false positive rate also appears to be a concave function with respect to this sequence. Therefore, the optimal choice of (d, s) can be obtained by finding the minimum point. The proposed algorithm to derive the optimal choice of k, d and s is listed in Figure 5. First we obtain the initial (d, s) using Eq. (17), and then starting from the initial (d, s), we obtain the optimal choice of (d, s) using the conjecture that the false positive rate is a concave function w.r.t. the (d, s) sequence described above. In addition, for each considered (d, s) pair, the optimal k value can be obtained by using the conjecture that the false positive rate is a concave function of k.

Note that the algorithm uses a two-level nested loop to search the optimal choice of (k, d, s). The outer loop searches the optimal choice of (d, s), and the inner loop searches the optimal k. These two loops are not exchangeable, because the false positive rate is not a concave function of k when the optimal choice of (d, s) is used for each k. The exact reason is unclear to us, but counter examples were found in our experiments.

2. Optimal configuration for a data set with numeric ranges

Procedure Optimal (k, d, s) configuration for a single numeric range
1. $s/d=m\times\ln 2/n$;
2. if (s/d)≥0.7, then (d, s)=(1, round(s/d));
3. else (d, s)=(round(s/d), 1);
4. For the initial (d, s), obtain the optimal choice of k by finding the minimum point of false positive rate w.r.t. k,
5. and store the optimal false positive rate.
6. On the list of (d, s) pairs (⋯, (2,1), (1,1), (1,2), (1,3),⋯), starting from the initial (d, s), find the optimal choice
7. of (d, s) that minimize the false positive rate.
8. //Note that for each (d, s) under consideration, its optimal choice of k can be obtained by finding the minimum
9. point of false positive rate w.r.t. k.
10. Return the optimal choice of (k, d, s) and the obtained minimum false positive rate.

Figure 5. Searching the optimal (k, d, s) configuration for a single numeric range.

Given a set of attributes, the domain size and the inserted range of each attribute, we need to find the optimal configurations to represent attributes so that the total false positive rate, which is defined as the summation of per attribute's false positive rate, is minimized. Let $\{A_1, A_2, \ldots, A_n\}$ denote the attribute set, R_i and n_i are the domain size and the inserted range size of attribute A_i, respectively. Note that when A_i is of atomic data type, $n_i = 1$. We assume that for all attributes, the number of hashed positions (i.e., k) of each key is the same. We develop an algorithm to derive the local-optimal parameter configurations, and this algorithm consists of two phases. Phase 1 determines the initial k value by utilizing the marginal utility theory of Economics, and Phase 2 minimizes the total false positive rate of all attributes through gradient descent search. Both phases are described in detail in the following.

A. Phase 1: determining the initial values of (k, d_i, s_i) through marginal utility theory

To represent a set of attributes of different scope sizes and different range sizes, we need to carefully allocate array positions to each attribute. From the marginal utility analysis of Economics, we know that utility is maximized when the consumer's budget is so allocated that the marginal utility to price ratio is equal for each good. This marginal utility theory can be applied to obtain the initial choices of d_i (the group size) and s_i (the number of hashed positions generated by each key) of each A_i. For a Bloom filter, the utility of each attribute can be represented by its false positive rate; the price is the number of hashed positions allocated for the attribute. The total budget is assumed to be $m$ln2, i.e., half of the array positions are expected to be set, and thus p =1/2. Since the false positive rate indicates the utility, the lower the better.

For an attribute of atomic data type (i.e., text string or integer), the false positive rate, denoted as f, can be expressed as follows.

$$f = (1 - p)^k \tag{1}(18)$$

To reduce the false positive rate, an extra array position can be allocated by increasing the

number of hashed positions (k) by 1. Assuming p is close to 1/2, the reduced false positive rate, denoted as f', can be approximated as in (19)

$$f' = (1-p)^{k+1} \approx \frac{1}{2} f \tag{18}$$

The marginal utility to price ratio can be approximated as

$$\frac{f'-f}{k+1-k} \approx -\frac{1}{2}(1-p)^k \approx -(\frac{1}{2})^{k+1} \tag{19}$$

In the following approximations for the false positive rate of a numeric range, we assume that n_i is small with respect to R_i (i.e., $n_i << R_i$) and p is close to 1/2.

(a) Case 1: considering attribute A_i whose $s_i = 1$ and $d_i > 1$

Such an attribute will be referred to as a grouping-overlapping attribute because both grouping and overlapping techniques are used. The false positive rate in (14) can be approximated by

$$\begin{aligned} f &= \frac{d_i - 1}{R_i - n_i} + \frac{2d_i}{R_i - n_i} \sum_{j=1}^{r} (1-p)^{js_i} + \frac{R_i - n_i - (d_i - 1) - 2rd_i}{R_i - n_i}(1-p)^k \\ &\approx \frac{3d_i - 1}{R_i} + \frac{R_i - n_i - (d_i - 1) - 2rd_i}{R_i - n_i}(1-p)^k \end{aligned} \tag{20}$$

From (11), the number of hashed positions generated by this attribute can be expressed as

$$b_i = \frac{n_i - 1}{d_i} + k \tag{21}$$

To reduce the false positive rate, extra array positions can be allocated by increasing d_i by 1. Thus the reduced false positive rate, f', can be approximated as follows.

$$f' \approx \frac{3d_i + 2}{R_i} + \frac{R_i - n_i - d_i - 2r(d_i + 1)}{R_i - n_i}(1-p)^k \tag{22}$$

The increased number of hashed positions allocated, b_i', can be expressed as follows.

$$b_i' = \frac{n_i - 1}{d_i - 1} + k \tag{23}$$

The marginal utility to price ratio can be approximated as follows.

$$\frac{f'-f}{b_i'-b_i} \approx -\frac{3d_i - 1}{R_i n_i} \tag{24}$$

In an optimally-configured Bloom filter, the marginal utility to price ratios for all attributes should be the same. From (20) and (25), we have

$$(\frac{1}{2})^{k+1} \approx \frac{3d_i - 1}{R_i n_i} \tag{25}$$

From (26), we can obtain

$$d_i \approx \sqrt{\frac{R_i n_i}{3}(\frac{1}{2})^{k+1}} \tag{26}$$

Eq. (27) indicates that the group size of a grouping-overlapping attribute, d_i, can be determined by the chosen k.

(b) Case 2: considering attribute A_i whose $d_i = 1$ and $s_i \geq 1$

Such an attribute will be referred to as an overlapping attribute because of using overlapping technique. The false positive rate in (10) can be approximated by

$$\begin{aligned} f &= \frac{2}{R_i - n_i}\sum_{j=1}^{r}(1-p)^{js_i} + \frac{R_i - n_i - 2r}{R_i - n_i}(1-p)^k \\ &\approx \frac{2(1-p)^{s_i}}{R_i} + \frac{R_i - n_i - 2r}{R_i - n_i}(1-p)^k \end{aligned} . \tag{27}$$

From (7), the number of hashed positions generated by this attribute can be expressed as follows.

$$b_i = (n_i - 1)s_i + k \tag{28}$$

Extra array positions can be allocated to this attribute by increasing s_i by 1. The reduced false positive rate, f', can be approximated as follows.

$$f' \approx \frac{2(1-p)^{s_i+1}}{R_i} + \frac{R_i - n_i - 2r}{R_i - n_i}(1-p)^k \tag{29}$$

The increased number of hashed positions allocated can be expressed as follows.

$$b_i' = (n_i - 1)(s_i + 1) + k \tag{30}$$

The marginal utility to price ratio can be expressed as follows.

$$\frac{f' - f}{b_i' - b_i} \approx -\frac{(1-p)^{s_i}}{R_i n_i} \tag{31}$$

For an optimally-configured Bloom filter, all attributes have the same marginal utility to price ratio. Thus from (20) and (32), we have

$$(\frac{1}{2})^{k+1} \approx \frac{(1/2)^{s_i}}{R_i n_i} \tag{32}$$

From (33), we further obtain

$$s_i \approx k + 1 - \frac{\ln(R_i n_i)}{\ln(2)} \quad , \tag{33}$$

Eq. (34) indicates that for an overlapping attribute, the number of hashed positions generated by each inserted key, s_i, can also be determined by the chosen k.

Note that no grouping attribute is considered, because grouping causes more false positive. Given k, we can determine whether attribute A_i is a grouping-overlapping or overlapping attribute from Eqs. (27) and (34). d_i and s_i can be obtained as follows.

$$d_i = \max(1, \text{round}\,(\sqrt{\frac{R_i n_i}{3}(\frac{1}{2})^{k+1}})) \tag{34}$$

$$s_i = \max(1, \text{round}(k + 1 - \frac{\ln(R_i n_i)}{\ln(2)})) \tag{35}$$

If d_i =1, it is an overlapping attribute; otherwise, it is a grouping-overlapping attribute. The algorithm that determines the initial k (denoted as k_{ini}) is shown in Figure 6. The initial k, k_{ini}, is chosen so that less than half of the array positions set to 1 with k_{ini} -1 hashed positions for each key, and more than half of the array positions set with k_{ini} hashed positions for each key.

B. Phase 2: minimizing the total false positive rate by gradient descent

In Phase 1, we obtain k_{ini}, d_i and s_i such that approximately half of the array positions will be set and the marginal utility to price ratio is about the same for all attributes. However, to minimize the total false rate of all attributes, we still need to determine the optimal choice of k, d_i and s_i. Let f_T denote the total false positive rate. For a given k, f_T may be further reduced by increasing or decreasing the number of hashed bits of the attributes. For each attribute, we consider the effects of increasing and decreasing its allocated hashed positions on f_T. Using gradient descent search, the hashed position re-allocation of the attribute that has

```
Procedure Initial value determination (m, R_i, n_i, k)
1. k = 0;   p = 0.5;
2. do
3. {  increase k by 1;
4.    b = 0;
5.    for all attribute A_i
6.       {  obtain d_i, s_i and b_i from Eqs. (35, 36, 22, 29);
7.          b = b + b_i ; }
8.  }while (b < m ln2)
9. return (k);
```

Figure 6. Phase 1 determines the initial k value.

```
Procedure Minimizing the total false positive rate (m, R_i, n_i, k_ini, d_i, s_i)
1.  f_opt = the number of attributes ;   //assume that f of all attributes =1
2.  for (k= k_ini-4; k<= k_ini+3;k++)
3.  {
4.   Given k, obtain (d_i, s_i) of all attributes using Eqs. (35, 36) ;
5.   Compute b, the total hashed positions of all attributes ;
6.   Compute p, the probability that an array position is set to 1 ;
7.   Compute f_T, the total false positive rate of all attributes ;
8.    Do until f_T cannot be reduced
9.    {
10.      For all non-atomic attributes
11.        check if f_T can be reduced by increasing or decreasing hashed positions ;
12.      Select the re-allocation that improves f_T the most per bit change ;
13.      Re-compute b, p, and f_T;
14.  }
15.  If f_T < f_opt
16.      {f_opt = f_T ; k_opt = k ; store (d_i, s_i) of all attributes;}
17.  }
```

Figure 7. Phase 2 minimizes the total false positive rate.

the largest improvement per bit change on f_T is chosen. Note that the improvement of f_T per bit change used in Phase 2 is similar to the marginal utility to price ratio used in Phase 1. However, the utility function used in Phase 2 is the total false positive rate; by contrast, that of Phase 1 is the false positive rate of the attribute under consideration. In addition, the value of p (the probability that an array position is set) is acutely computed in Phase 2, while it is assumed to be 1/2 in Phase 1. Since we cannot be sure of the optimal choice of k, given k_{ini} obtained in Phase 1, the gradient search for the optimal choice of d_i and s_i is performed for all k's where k_{ini} -4 ≤ k ≤ k_{ini} +3. The detailed Phase 2 algorithm is shown in Figure 7.

V. Simulation and Analytic Results

To validate the analytic model presented in Section III and to evaluate the performance of the configuration algorithms presented in Section IV, we developed a simulation program that computes the expected false positive rate when a single numeric range is inserted in a grouping and/or overlapping Bloom filter. The inputs of the simulation program include the attribute name, the scope of the attribute and the size of the inserted range. The program literally computes the average false positive rate for all the possible inserted ranges of a given size. For example, given attribute Age with scope [0..120] and an inserted range of size 10, the program can compute the average false positive rate for all possible inserted ranges in {[0..9], [1..10], [2..11], …, [111..120]}. However, the analytic models presented in Section III assume that overlapping false positives exit on both ends of each inserted range. Therefore, our simulation program ignores the inserted ranges that do not allow all possible overlapping false positives on either end. For an example attribute, Age, if (k, d, s) = (8, 10, 2), the program ignores inserted ranges {[0..9], [1..10], [2..11], [109..118], [110..119],[111..120]}, and only considers {[3..12], [4..13], [5..14], …, [108..117]}, so that on each end of the inserted ranges considered, there are exactly 3 numbers (i.e., $\lceil k/s \rceil - 1$ shown in Eq. (13)) that may cause overlapping false positives. For any given inserted range, the simulation

program first generates the Bloom filter, checks each number outside the inserted range to determine whether it is a false positive or not, and then calculates the average false positive rate.

The simulation program uses SHA-512("attribute_name:attribute_value") to generate the hashed array positions for each key to be inserted. The 512-bit digest of SHA-512 is partitioned into 32 16-bit words. When *s* hashed array positions are needed, the first *s* words are used; each word modulo 512 (*m*) generates an array position.

Figure 8 depicts the false positive rates of a grouping-overlapping Bloom filter as the inserted range increases from 4 to 60, the scope is [0,..,120] and *k* is 10. The same optimal choice of (*d*, *s*) is used for each inserted range. The analytic model does not consider the attribute name. The simulation results are obtained through using attribute name Age, while the simulation average results are obtained from averaging the false positive rates of 1000 random attribute names. The analytic and simulation results for attribute Age are consistent for most inserted ranges, but there exists significant discrepancy when the inserted range size is near 52. The reason is the analytic models assume that the adopted hash functions are

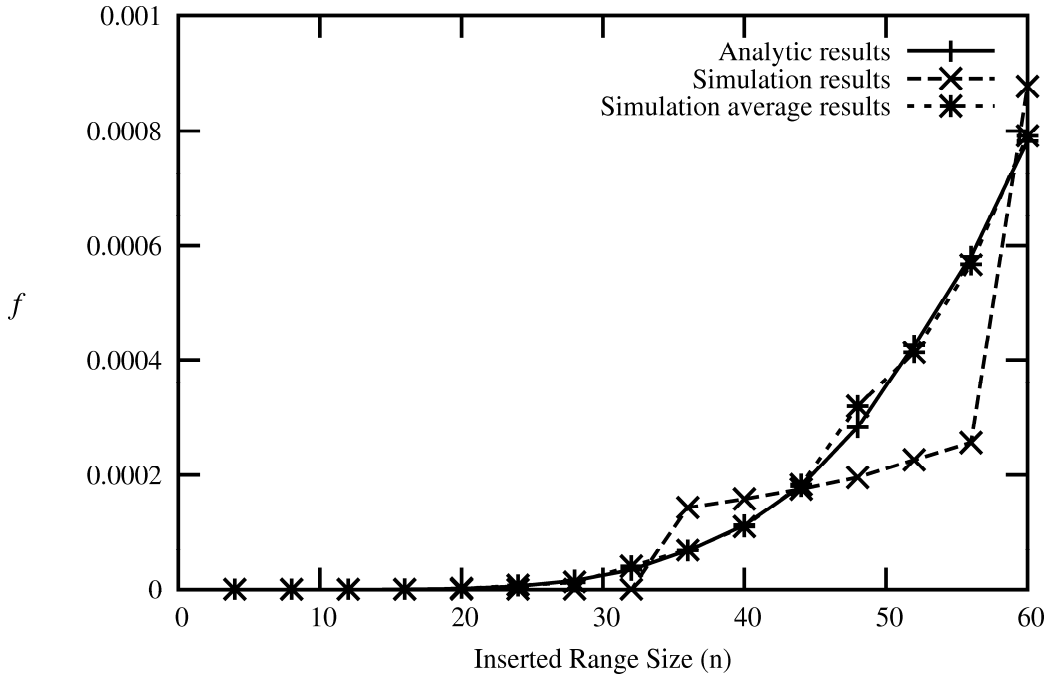

Figure 8. The analytic and simulation results.

perfectly random. However, this may not be true for a specific attribute name, “Age” in this example, which leads to a smaller false positive rate. On the other hand, the analytic and average simulation results are consistent for all sizes of inserted ranges. This verifies the correctness of our analytic model, and indicates our analytic model produces the average false positive rates. To represent the average case and to speed up the comparison, for the experiments hereafter, we will only use the analytic model.

1. Scenario 1: a single numeric range

For all experiments on representing a single numeric range hereafter, the scope size of the attribute R is 10000, and the size of Bloom filter is 512 bits (i.e., m = 512). Figure 9 plots the false positive rates of four schemes when a single numeric range is inserted and the size of the inserted numeric range increases from 10 to 1200. The optimal choice of (k, d, s) is used for each inserted range under each Bloom filter scheme. The results indicate that for a given inserted range, the traditional Bloom filter has the largest false positive rate. The false positive rates of both the traditional and the grouping Bloom filters increase rapidly when the size of the inserted range (n) is larger than 20. When n = 22, the grouping Bloom filter starts

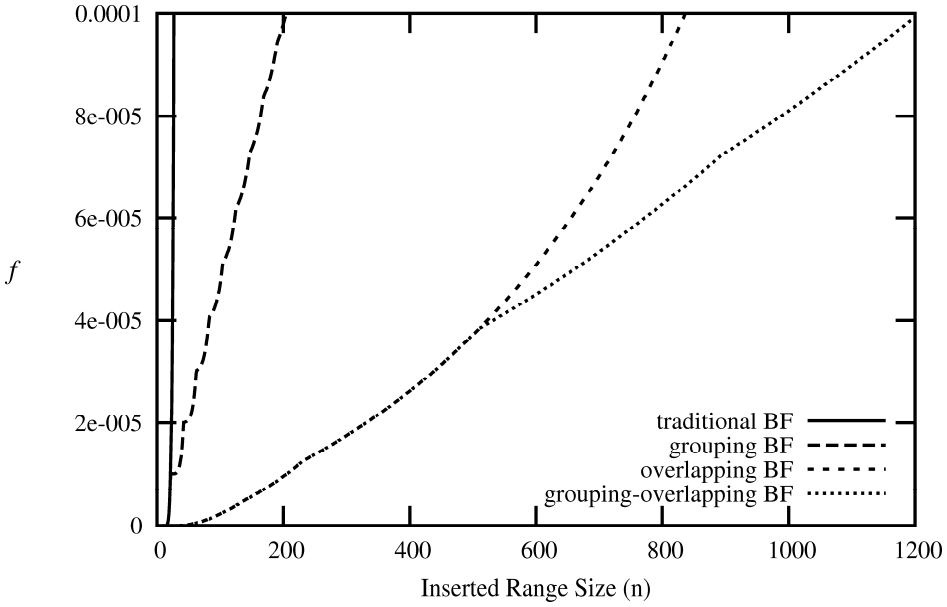

Figure 9. The impact of the inserted range size on false positive rate.

grouping the inserted range. Although the grouping Bloom filter provides significant improvements over the traditional one, it still performs much worse than both schemes adopting overlapping technique. When n is less than 510, the overlapping and the grouping-overlapping Bloom filters are the same, i.e., only overlapping is used. However, when n is larger than 510, both overlapping and grouping need to be used, and the grouping-overlapping Bloom filter provides the smallest false positive rate among four schemes. The results indicate that overlapping technique is useful for all sizes of inserted ranges, and grouping is only applicable when the size of the inserted range is very large. The results also explain why a grouping-overlapping Bloom filter with (d, s) = (2, 7) has a higher false positive rate than one with (d, s) = (1, 3) or (1, 4).

Let a_1 denote the expected number of array positions that are set to 1. a_1 can be derived as follows.

$$a_1 = m(1-(1-\frac{1}{m})^h), \tag{36}$$

where h is the number of hashed positions generated by an inserted range for a Bloom filter.

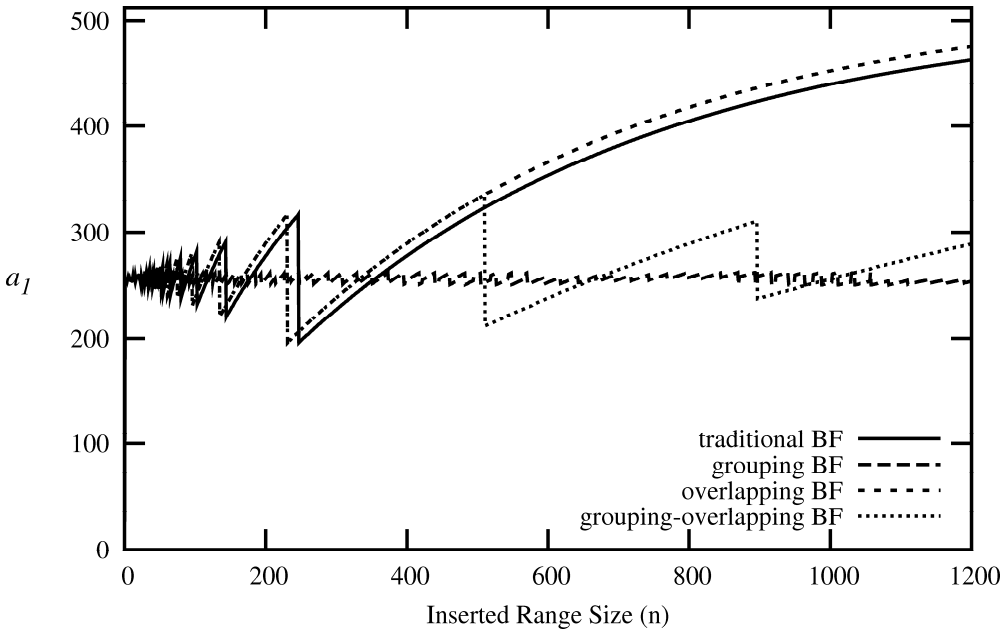

Figure 10. The expected number of array positions that are set to 1 for all schemes.

Figure 10 plots a_1 of each Bloom filter scheme, upon varying inserted range size from 1 to 1200, and R being 10000. We expect that about half (i.e., 256) of the array positions are set to 1. The grouping Bloom filter has the slightest variations in a_1, as n increases. On the other hand, the others have large variations. It is interesting that a_1 of those schemes are of zigzag curves. Each drop of the a_1 curves indicates a decrement of k for the traditional Bloom filter, and a decrement of s for both the overlapping and the grouping-overlapping Bloom filters for $n < 250$. When $n > 250$, $k = 1$ for the traditional Bloom filter, and thus k cannot be decreased further. In addition, $s = 1$ for the overlapping Bloom filter, and cannot be reduced further. As a result, after that, a_1 of both schemes increases steadily as n increases. On the other hand, the grouping-overlapping Bloom filter starts grouping by increasing d by 1 when n=511 and 897. As a result, a_1 is kept close to $m/2$ (256). The results are consistent with the results in Figure 9; the false positive rate of the grouping-overlapping schemes increases slower than that of the overlapping scheme as n increases.

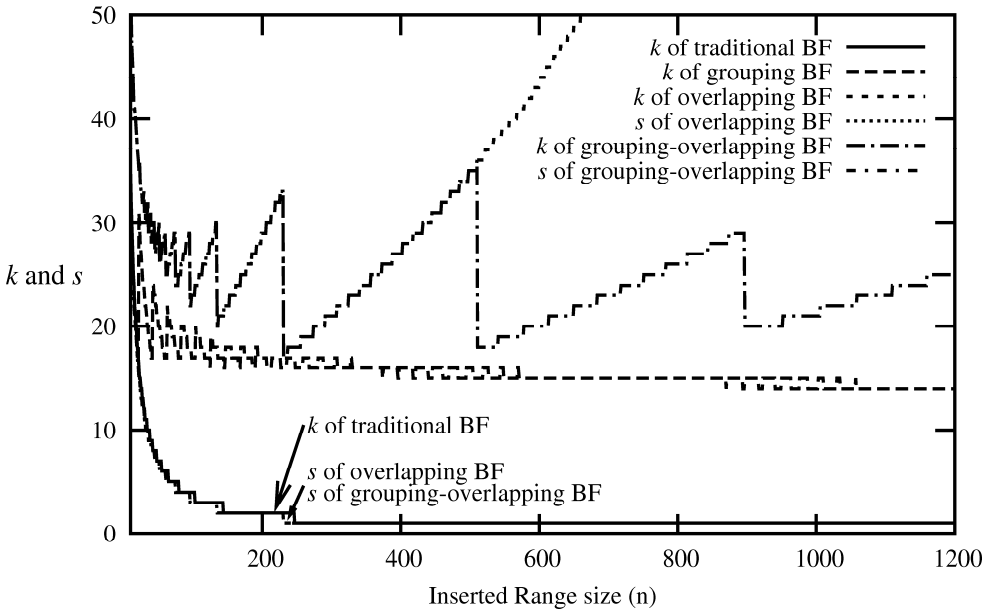

Figure 11. The optimal values of k and s for all schemes.

Figure 11 plots the values of k and s of all schemes and Figure 12 plots the values of d for the grouping and grouping-overlapping schemes upon varying n value from 5 to 1200.

The results show that k of the traditional Bloom filter and s of the overlapping and grouping-overlapping Bloom filter drop rapidly and almost at the same pace, as n increases. This implies that the three schemes set about the same number of array positions; this is consistent with the results of Figure 10. In addition, we observe that the overlapping and the grouping-overlapping Bloom filters are the same when $n < 510$. Moreover, when $n > 510$, k of the overlapping Bloom filter keeps increasing, while k of the grouping-overlapping Bloom filter makes a steep dive (shown in Figure 11) and the corresponding d value of the grouping-overlapping Bloom filter increases by 1 at the same time (as shown in Figure 12). On the other hand, the k values of the grouping scheme fluctuate substantially when n is small, and then becomes stable as n increases. For all sizes of inserted range, the k value of grouping scheme is less than that of the overlapping and grouping-overlapping schemes. The results in Figure 12 indicate that d of the grouping and grouping-overlapping schemes are both staircase functions. However, d of the grouping scheme increases at a much faster pace than that of the grouping-overlapping scheme. When $n = 1200$, the d value of the grouping-overlapping scheme is only 2, while that of the grouping scheme is 50. This indicates that the grouping false positives are much more common in a grouping Bloom filter.

We further investigate the performance of false positive rate upon fixed k value. This constraint is practical because the size of the inserted range is usually not fixed. By using a constant k, we set the desired false positive rate below $(1/2)^k$ when less than half of the array positions are set to 1. Figure 13 plots the false positive rates of each scheme as the inserted range increases, and k is fixed at 10. In other words, the desired false positive rate is below $(1/2)^{10} \approx 0.001$. The optimal choices of d and s are used for each inserted range under each Bloom filter scheme. The results indicate that a traditional Bloom filter cannot satisfy the desired false positive rate when the size of a stored numeric range is larger than 60. In contrast, meeting the requirement of the false positive rate, the grouping Bloom filter can

store a numeric range of size up to 240, the overlapping Bloom filter up to 340, and the grouping-overlapping Bloom filter up to 1000. Note that although the false positive rate of an overlapping Bloom filter is very low when *n* is less than 240, it increases more rapidly than that of a grouping Bloom filter afterwards. When *n* is larger than 360, its false positive rate surpasses that of the grouping Bloom filer. In summary, as the size of the inserted range increases, we suggest utilizing only overlapping technique first, and then both overlapping and grouping techniques.

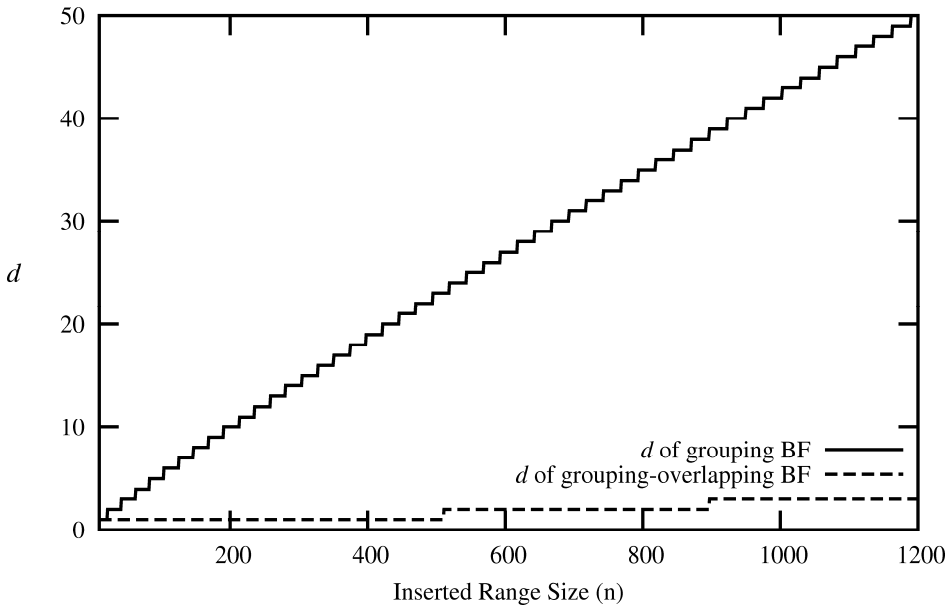

Figure 12 The optimal values of *d* for the grouping and grouping-overlapping schemes.

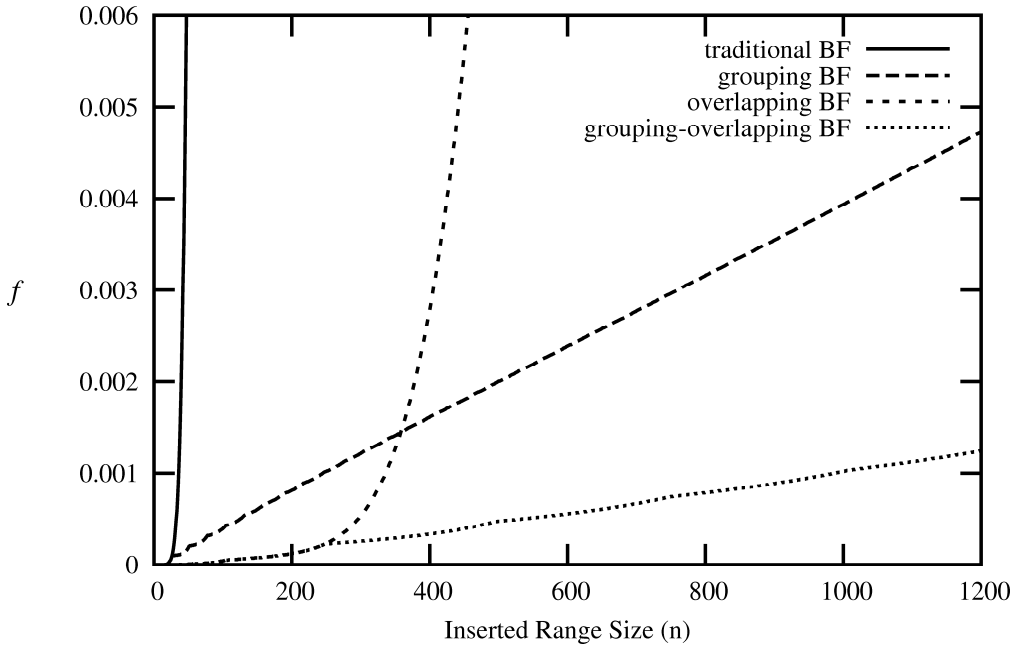

Figure 13. The false positive rates of all scheme when k is fixed (=10).

2. Representing a data set with numeric ranges

To evaluate the performance of a grouping-overlapping Bloom filter for multiple attribute range queries, we consider an example attribute set listed in Table 1. There are five attributes of atomic data type (text string) and five attributes of numeric range. The attribute names, scope sizes, and inserted range sizes are also listed in Table 1. Given the attributes, the Phase 1 configuration algorithm obtains the initial k value, k_{ini} = 15. Through the Phase 2 algorithm, the optimal configurations and minimal false positive rates for all k's, where k_{ini} -4 $\leq k \leq k_{ini}$ +3, are obtained. Table 2 summarizes the expected numbers of array positions set to 1 (i.e., a_1), and the total false positive rates (f_T) for various k's. The results indicate a_1 increases as k increases, and about half of the array positions are set when $k = k_{ini}$-1. In addition, f_T appears to be a concave function of k, and the optimal k is the k_{ini} obtained in Phase 1. The difference in f_T for all k's is insignificant. This implies the optimal choice of k may not be as important as the optimal choice of (d, s) for each attribute.

Table 1. The attributes used in multi-attribute experiments.

Attribute Name	Data Type and Scope	Size of inserted range
First Name	Text string	—
Last Name	Text string	—
University	Text string	—
Hobby	Text string	—
Vocation	Text string	—
Age	Integer [1,..,120]	10
Year	Integer [1900,..,2100]	20
Income	Integer [0,..,5000]	500
Longitude	Integer [-1800000,..,1800000]	100
Latitude	Integer [-900000,..,900000]	100

Table 2. The expected number of array positions set to 1, and the total false positive rates.

	k = 11 (k_{ini}-4)	k = 12 (k_{ini}-3)	k = 13 (k_{ini}-2)	k = 14 (k_{ini}-1)	k = 15 (k_{ini})	k = 16 (k_{ini}+1)	k = 17 (k_{ini}+2)	k = 17 (k_{ini}+3)
a_1	214	232	237	261	270	275	280	284
f_T	5.818E-3	5.652E-3	5.602E-3	5.549E-3	5.496E-3	5.503E-3	5.589E-3	5.727E-3

Table 3 displays the detailed configurations of the grouping-overlapping Bloom filter and the false positive rate of each attribute. The results indicate that the false positive rates of atomic data type decrease as k increases. On the other hand, the false positive rates of numeric ranges are in the form of irregular patterns. They may not be a concave function of k; attribute Age is such an example. The configurations to minimize the false positive rates per-attribute basis and all-attributes basis are not exactly the same. For example, the minimum total false positive rate occurs at k = 15, while the minimum false positive rate of attribute Longitude at k=16. Note that the number of array positions allocated for each attribute (b_i) increases as k increases. In addition, attributes with larger false positive rates have more array positions. For example, among all attributes, Income accounts for 62% (3.4/5.496≈0.619) of the total false positive rate and obtains the largest number of array positions. The results show that the grouping-overlapping Bloom filter provides very low

false positive rates for all attributes and this is not attainable for the traditional Bloom filter.

Table 3. The detailed configurations and false positive rates of all attributes.

Attribute Name	k = 13 (k_{ini}-2)			k = 14 (k_{ini}-1)			k = 15 (k_{ini})			k = 16 (k_{ini}+1)		
	d, s	b_i	f_i	d, s	b_i	f_i	d, s	b_i	f_i	d, s	b_i	f_i
First Name	-	13	4.5E-5	-	14	7.8E-5	-	15	6.7E-5	-	16	4.7E-5
Last Name	-	13	4.5E-5	-	14	7.8E-5	-	15	6.7E-5	-	16	4.7E-5
University	-	13	4.5E-5	-	14	7.8E-5	-	15	6.7E-5	-	16	4.7E-5
Hobby	-	13	4.5E-5	-	14	7.8E-5	-	15	6.7E-5	-	16	4.7E-5
Vocation	-	13	4.5E-5	-	14	7.8E-5	-	15	6.7E-5	-	16	4.7E-5
Age	1,5	58	4.4E-4	1,5	59	7.2E-4	1,6	69	4.6E-4	1,6	70	4.9E-4
Year	1,3	70	1.3E-3	1,4	90	8.7E-4	1,4	91	9.9E-4	1,4	92	1.0E-3
Income	6,1	96	3.5E-3	5,1	114	3.3E-3	5,1	115	3.4E-3	5,1	116	3.5E-3
Longitude	70,1	14	9.8E-5	70,1	15	1.4E-4	65,1	17	1.3E-4	61,1	18	1.0E-4
Latitude	49,1	15	1.2E-4	50,1	16	1.6E-4	46,1	17	1.5E-4	43,1	18	1.3E-4

We expect that when k=k_{ini}, more than half of the array positions are set to 1, and when k=k_{ini}-1, less than half. Therefore, we speculated that both cases are likely to be the optimal choice of k. To verify our speculation, we randomly generate 1000 attribute sets; each set contains 10 numeric range attributes. The scopes of the attributes are uniformly distributed in [100, .., 1000000], and the inserted range sizes are uniformly distributed in [0.001, 01]×R_i (the corresponding scope size). The distribution of the optimal k values is shown in Table 4. For 781 out of 1000 samples, k_{ini} is the optimal choice of k, and k_{ini}-1 is the optimal choice for another 216 samples. Only three samples' optimal k choices are not k_{ini} or k_{ini}-1. The results suggest that k_{ini} is a very good initial choice, and the optimal configuration tends to set more than half of the array positions to 1. Table 5 is the distribution of a_1, the expected number of array positions that are set to 1 by the optimal configurations. The results indicate that slightly more than half of the array positions are set to 1 for the optimally configured grouping-overlapping Bloom filter.

Table 4. The distribution of the optimal k.

The optimal k	k_{ini}-3	k_{ini}-2	k_{ini}-1	k_{ini}	k_{ini}+1	k_{ini}+2
Occurrences	0	1	216	781	2	0

Table 5 The distribution of the expected number of array positions set to 1.

	≤ 254	[255,..,259]	[260,..,264]	[265,..,269]	[270,..,274]	[275,..,279]	≥ 280
Occurrences	2	1	77	502	361	56	1

VI. Conclusions

Traditional Bloom filter is space-inefficient to represent data sets that contain large ranges of numeric data. In this report, we presented grouping and overlapping Bloom filter schemes for representing a data set with numeric ranges. In addition, we have developed algorithms based on the marginal utility theory and gradient descent to obtain the optimal configurations of the modified Bloom filter schemes. Numeric analysis and computer simulations have been conducted to show the effectiveness of our schemes and configuration algorithms. Our experiments showed that the overlapping technique can be used for all sizes of numeric ranges, while the grouping technique should only be applied for large numeric ranges. The numeric results also showed that the grouping-overlapping Bloom filter provides very low false positive rates for multi-attribute range queries, which are not attainable for the traditional Bloom filter. Moreover, the optimally-configured grouping-overlapping Bloom filter set slightly more than half of the array positions.

In this report, we assume the number of hashed positions, k, of an inserted key of each attribute is the same. It is possible that different k for different attributes may result in a lower overall false positive rate. In addition, the conjectures we made in searching the optimal choice of (k, d, s) for a numeric range may need a formal proof. The optimal configuration of a grouping-overlapping Bloom filter for a data set with numeric ranges obtained from our algorithm may not be the optimum configuration, but only a local optimum.

References

[1] B. H. Bloom, “Space/Time Trade-offs in Hash Coding with Allowable Errors,” *Communications of the ACM*, vol. 13, no. 7, pp. 422-426, 1970.

[2] A. Broder and M. Mitzenmacher, “Network applications of Bloom filters: a survey”, *Internet Mathematics*, vol. 1. no. 4, pp. 485-509, 2005.

[3] A. Rousskov and D. Wessels, “Cache digests,” *Computer Networks and ISDN Systems*, vol. 30, pp. 2155-2168, 1998.

[4] J. Kubiatowicz, et al., “OceanStore: An architecture for global-scale persistent storage,” *Proceeedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS 2000), pp. 190–201, 2000.

[5] P. Reynolds and A. Vahdat, “Efficient peer-to-peer keyword searching,” *ACM/IFP/USENIX Int’l Middleware Conference*, June 16–20, 2003.

[6] W.C. Feng, D.D. Kandlur, D. Saha, and K.G. Shin, “Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness,” *Proceedings of the IEEE INFOCOM*, pp. 1520–1529, 2001.

[7] A. Broder, M. Mitzenmacher, “Using multiple hash functions to improve IP lookups,” *Proceedings of the IEEE INFOCOM*, pp. 1454–1463, 2001.

[8] S. Dharmapurikar, P. Krishnamurthy, D.E. Taylor, “Longest Prefix Matching Using Bloom Filters,” *Proceedings of the ACM SIGCOMM*, pp. 201–212, 2003.

[9] A. Kumar, J. Xu, J. Wang, O. Spatschek, and L. Li, “Space-Code Bloom filter for efficient per-flow traffic measurement,” *Proceedings of the IEEE INFOCOM*, pp.1762-1773, 2004.

[10] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, “Summary cache: a scalable wide-area Web cache sharing protocol,” *IEEE/ACM Transactions on Networking*, vol. 8, no 3, pp. 281-293, 2000.

[11] M. Mitzenmacher, “Compressed Bloom filters”, *IEEE/ACM Transactions on Networking*, vol. 10, no. 5, pp. 604-612, 2002.

[12] D. Guo, J. Wu, H.H. Chen, and X.J. Luo, “Theory and Network Application of Dynamic Bloom Filters,” *Proceedings of the IEEE INFOCOM*, 2006.

[13] M. Cai, M. Frank, J. Chen, P. Szekely, “MAAN: A Multi-Attribute Addressable Network for Grid Information Services,” *Journal of Grid Computing*, vol. 2, pp. 3–14, 2004.

[14] C. Schmidt, and M. Parashar, "Enabling Flexible Queries with Guarantees in P2P Systems," *IEEE Internet Computing*, vol. 8, no. 3, pp. 19-26, 2004.