Efficient Web Profiling by Time -Decaying Bloom Filters

Kai CHENG♥ Mizuho IWAIHARA♦
Limin XIANG♠ Kazuo USHIJIMA♠

We propose a space-efficient data structure, called Time-decaying Bloom Filters (TBF) to maintain time-sensitive profiles of the web. TBF extends the standard Bloom Filters (for approximate membership queries) by replacing the bit-vector with an array of counters, whose values decay periodically with time elapsing. Due to the skewed distribution of web usage, only a small fraction of web contents are frequently visited. To avoid allocating larger counters to small values, we propose a dynamic approach for counter management, where only *heavy hitters* are monitored by larger counters. The preliminary experiments show that the optimized TBF achieves considerable improvement on space usage while providing the same results of hit frequency profile.

1. Introduction

The web is a vest repository of online information. The huge scale of the web makes it a big challenge to retrieve high quality and most relevant information efficiently. A promising way to meet the challenge is to tailor the specific information services by exploiting various profiles of the web. For example, citation profile indicates the authority of web documents, which has been widely used to filter out low quality web contents, e.g., Google. It has recently been noted that search services can also be improved by incorporating hit frequency of web pages while computing the rank of search results. Since hit frequency indicates the popularity of web pages from the readers' perspective, it is expected to exploit as a guide for less experienced web users [7].

However, most previous work has focused on profiles derived from *static* properties of the web, e.g., term /document frequency, link structures and so on. *Time-sensitive* profiles, such as hit frequency, are rarely used in serious applications due to the following challenges. The primary challenge in maintaining time-sensitive profiles lies in the fact that the volume of web contents is huge but there are only limited resources (memory space and CPU time) available. The current size of the web is expected to be several billions and popular portal sites often see hundreds of millions requests per day. The challenge also comes from the time-sensitivity of web usage due to the fast change of user interests. Thus recent data more accurately represent the real state of the web usage than past one. It is important to deal with such time-sensitivity of web usage.

The brute-force approach to maintaining hit frequency is to use as many counters as possible for all available web contents. This can only be applicable in small-scale web profiling, but is unrealistic for large-scale applications, such as busy portal sites, or general-purpose search services due to the limited memory space and CPU time available.

Bloom Filters are space-efficient data structures that maintain a very compact inventory of the underlying data, supporting membership queries over a given data set [1].The space requirements of Bloom filters fall significantly below the information theoretic lower bounds for error-free data structures. This efficiency is at the cost of a small false positive rate (items not in the set have a small constant probability of being listed as in the set), but have no false negatives (items in the set are always recognized as being in the set). Bloom filters are widely used in practice when storage is at a premium and an occasional false positive is tolerable.

The standard Bloom Filters use a bit-vector to hold information about the underlying data set. Initially, all bits in the bit-vector are turned off. Each item in the set is hashed into several locations of the bit-vector using deferent hash functions. Bits at these locations are then turned on. In order to find whether an item is a member of the set, we first compute the locations to which the item is mapped according to the same hash functions. We then check if all bits at these locations are on. If so, the answer is *yes*; otherwise answer *no*. As a bit can be set by other items due to the hash collisions, thus false positives are possible.

Standard Bloom Filters, although useful, have several limitations. First, they do not support deletes because simply turning off the corresponding bits may introduce new false negative errors (some bits of other items, although still in the set, may be turned off). Second, they cannot deal with multi-set (set with duplicates) to return multiplicities of items. Many applications, e.g., iceberg queries [2], data mining, and data stream management, such multiplicity information is crucial.

Several improvements have been made over the original Bloom Filter. In [3], Li, F. et al proposed *Counting Bloom Filters (CBF)*, where a counter has been attached to each bit in the array to count the number of items mapped to that location. Thus, it can support deletions in a set by decrementing the corresponding counters by 1. To maintain the compactness of the structure, these counters were limited to 4 bits.

In [4], Cohen, S. et al proposed a sophisticated implementation for CBF, called, Spectral Bloom Filters (SBF). It introduces efficient schemes for counter updates, such as Minimum Increase (MI), Recurring Minimum (RM) that try to reduce false positive rate. In addition, it develops variable length string access methods for compact representation of the counters, which thus in principle allow counters of arbitrary length. However, to the best of our knowledge, none of the proposed Bloom Filter improvements supports time-sensitive counting of the multiplicity.

Time-sensitivity is important since in many traditional

♥ Regular Member Faculty of Information Science, Kyushu Sangyo University. chengk@is.kyusan-u.ac.jp
♦ Regular Member Graduate School of Informatics, Kyoto University. iwaihara@i.kyoto-u.ac.jp
♠ Faculty of Information Science, Kyushu Sangyo University. {xiang,ushijima}@is.kyusan-u.ac.jp

and emerging applications, data streams play an increasingly important role, e.g., web tracking and personalization, medical monitoring, sensor databases, and financial monitoring. In most of these applications, the goal is to make decisions based on the statistics or models gathered over the recently observed data. For example, one might be interested in gathering statistics about web sites visited in recent days. In most case, we have to maintain these statistics continuously.

In this article, we propose Time-Decaying Bloom Filters (TBF), a novel improvement over the current Bloom Filter variants. We address the challenges in maintaining time-sensitive frequency counts over data streams in general, and web usage data in particular. First, we propose a basic scheme, called basic TBF, for maintaining time-decaying aggregates over data streams. We then propose optimizations to the basic scheme by leveraging skewed distribution of web usage where only a small part of web sites get most hits. The optimized scheme, called exponential TBF, consists of a series of basic TBFs as components, each responsible for a small group of bits in a large counter. Since only a few "large" items will be added in the component TBFs that take care of higher significant bits of large counters, it is quite efficient.

2. Problem Definition

Following the notations in [5], we formalize the problem as follows. A stream S consists of a sequence of N item occurrences with time-stamps:

$$S = \{(e_1,t_1),(e_2,t_2),\cdots,(e_N,t_N)\}$$

Each item occurrence e_i is drawn from the universe U, Arbitrary repetition of item occurrences in streams is allowed. In the following, we use q, or e with or without subscripts to denote an item. Let f_e be the frequency count of item e in S at the current time. Let $g(x)$ be an non-increasing real-valued *decay function* [6]. A time period of T time units is referred to as an *epoch*.

Our goal is to give, at any time instance, an estimate $estimate(f_e)$ of f_e such that

1. $estimate(f_e) \geq f_e$, i.e., $estimate(f_e)$ never undercounts the occurrences of item e.
2. Error of the estimate $estimate(f_e)$ is bounded to an allowable level.
3. The space complexity is independent of the stream length.
4. $O(1)$ time complexity for adding an item into the data structure, and/or answering a query of frequency count for an item.

3. Time-Decaying Bloom Filters

In this section, we describe the basic form of Time-decaying Bloom Filters (TBF). A TBF is a counting Bloom filter (Bloom filter with the bit-vector replaced by an array of counters) whose counters are decayed with time elapsing.

3.1 Basic Time-Decaying Bloom Filters (B-TBF)

Given a set of independent hash functions, $h_1,\cdots,h_k$, from objects to [1..m]; an array of counters, $C_1,\cdots,C_m$,. Let Initially, all counters are reset to 0. When an item, say q, occurs, increment each of the counters $C_{h1(q)},\cdots,C_{hk(q)}$. When T time units elapsed, decay all counters by applying $g(x)$ to them.

Algorithm for Basic TBF Management

Input:

Data stream $S = \{(e_1,t_1),(e_2,t_2),\cdots,(e_N,t_N)\}$, t_i is the time instance while item e_i occurs ;

Data Structures:

An array of m counters: $\langle C_1, C_2, \cdots, C_m \rangle$

Output:

Estimate of the decayed frequency count for each item

Initialize:

for (i = 1; i≤ m){ C_i = 0; }

Increment (on arrival of a new item q):

for (j = 1; j≤k; j++) $C_{hj(q)}$ ++;

Decay (on start of a new epoch):

for (i = 1; i≤ m; i++) {$C_i = \lambda \cdot C_i$;}

Query(q):

estimate(C(q))=(1-λ) · Min{$C_{h1(q)}$, $C_{h2(q)}$, ⋯ ,$C_{hk(q)}$};

Return estimate(C(q))

3.2 Time-Decaying Counters

Dependent on the form of $g(x)$, the last operation can be quite complicated. In [6], there is an abundance of decay functions, e.g., exponential decay, sliding window decay, polynomial decay, polyexponential decay and chordal and polygonal decay. In this paper, we focus on exponential decay, the most commonly used decay function,

$$g(x) = \lambda^x, \quad [0,1]$$

In this case, a time-decaying counter can be defined as,

$$f_e = \sum_{(e,t_i)\in S} \lambda^{\left(\frac{t_{now}-t_i}{T}\right)} \qquad (1)$$

where t_{now} denotes the current time, and λ and T are user-defined parameters. The parameter $\lambda \in [0,1]$, called *exponential decaying factor* or simply *decaying factor*, controls the speed of exponential decaying. The parameter $T > 0$ controls the frequency with which the exponential decaying takes place. In other words, it controls the granularity of time-sensitivity.

Time-decaying counter defined in (1) can be maintained very efficiently in the following way. Given x_t, the net value obtained at time instance t after a period of T, the exponentially decayed count value can be recursively defined as

$$y_t = (1-\lambda)\cdot x_t + \lambda \cdot y_{t-1}, \quad \lambda \in [0,1] \qquad (2)$$

λ is called an *exponentially decaying factor* or simply *decaying factor*. For $\lambda<1$, let $y'_t = y_t/(1-\lambda)$, we then have the following form.

$$y'_t = x_t + \lambda \cdot y'_{t-1}, \quad \lambda \in [0,1) \qquad (3)$$

Using the above equation, we can increment a counter directly after the decay function was applied, i.e., $\lambda \cdot y'_{t-1}$. Because y'_t is available, the result of y_t can be obtained by the following equation.

$$y_t = (1-\lambda)y'_t \qquad (4)$$

3.3 Problems

There are a few problems with the basic TBFs, particularly in the web profiling context. Among others, using uniform-sized counters for all items is the most severe one. The usage of web is well known to be quite biased, with a

small fraction of popular sites getting very high hits, while the rest and also the majority are rarely used. As the values of counters vary significantly from very small values for the many "cold" pages to thousands of hits for a few "hot" pages it is not suitable to allocate the bits to count each of these items. Fig. 1 shows the typical distribution of web usage, where more than *85%* of web pages share less than *10%* visits, while about 10% of popular pages receive up to 90% accesses. Thus, *it is profitable to adaptively allocate non-equally-sized counters for different items.*

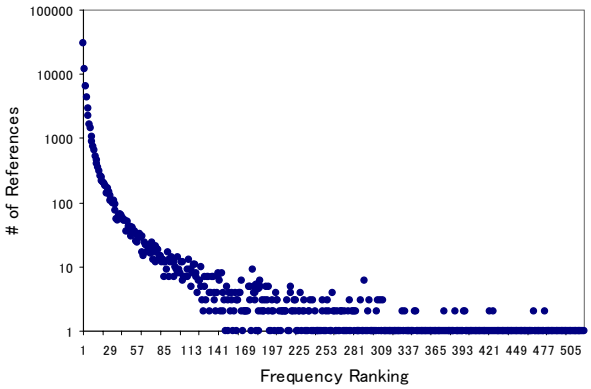

Fig. 1 Typical Skewed Distribution of Web References

4. Exploiting Skewed Distributions

4.1 Data Structures

To avoid allocating large counters for many “cold” pages, we optimize the basic TBF using a *dynamic representation of large counters*. The data structures include:

1. A standard Bloom Filter, BF(m,k,n)
2. A basic TBF, TBF1 with *k* pair-wise independent hash functions and *m* small counters
3. A free counter pool（FCP） with m/2 free counters
4. A lookup table(LT)

First, a standard Bloom Filters is used to enable quick membership query. The basic TBF with small counters hold frequency counts for the many “cold” items. For few “hot” items, extra counters can be allocated dynamically when necessary. When a small counter in the TBF gets overflowed, we allocate a new counter from the *free counter pool* (FCP) for the carried digits. A lookup table (LT) maintains the bi-directional links between the overflowed counters and the extra counters. A counter in the TBF can extend to a linked counter list to represent much larger values.

To insert a new item *q* into the optimized TBF filter, we simply insert the item into TBF1, incrementing each of its counters $h_{1,i}$ ($i=1,\cdots,k$) by 1 unless the counter becomes overflow. If ALL these counters get overflowed, and if there is an extra counter for this TBF1 counter, increment that counter by 1; otherwise allocate an extra counters for the TBF1 counter. Repeat the process if a extra counter gets overflow too.

To query for the frequency count of an item, say, *q*, we first check if *q* is recorded in BF0. If not, return *0*; otherwise, we check the TBF1 and the lookup table to constructing the whole counter for *q* from higher significant bits to lower ones. In Fig.2, *q* is recorded in the first classic BF, and there are 2 extra counters, holding 01 and 02 respectively. Assuming all are 4-bit counters, the frequency count for *q* by the TBF1 counter $C_{h12(q)}$ is 256*1+2*16 + 0 = 288.

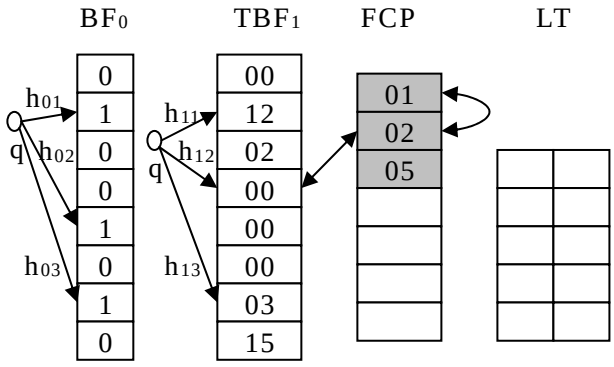

Fig. 2 Optimized TBF: Allocates Additional Counters for Few Hot Items from Free Counter Pool (FCP)

4.2 Optimized TBF (O-TBF)

Based on the above described data structures, we now give the algorithm for management of time-decaying hit frequencies with dynamic counter allocation.

Input:

Data stream $S=\{(e_1,t_1),(e_2,t_2),\cdots,(e_N,t_N)\}$, t_i is the time instance while item e_i occurs ;

Output:

Estimate of the decayed frequency count for each item

Initialize:

for ($i = 1; i \le m; i$++) $C_i \Leftarrow 0$; Reset lookup table *LT*

Increment (on arrival of a new item q):

for (j =1; j≤k; j++)
if ($C_{hj(q)}$ overflows) {
Requires a counter C_{new} from free counter pool
Register C_{new} and the associated $C_{hj(q)}$ in *LT*
}
$C_{hj(q)}$++

Decay (on start of a new epoch):

for (i =1; i≤ m; i++){
Assemble all linked counters into C_i
$C_i = \lambda \cdot C_i$
Free *0*-valued sub-counters in *Ci* at high end.

Query (q):

Return $estimate(C(q)) = (1-\lambda) \cdot Min\{C_{h1(q)},\cdots,C_{hk(q)}\}$

The exponential decaying operations take place whenever an epoch starts. We have to update each of TBF1 counters and the associated extra counters. First, collecting all bits of the whole counter (bits of TBF1 counters themselves as well as bits in the extra counters), say the "new" counter is C_i. Then compute $\lambda \cdot C_i$. Then, allocate lower 4 bits to the same counter in TBF1. The left may be written back to one or more extra counters, depending on the decaying results. If no value to write back, the extra counters will be returned to the free counters list.

4.3 Analysis

According to Section 2 the data stream S has N item occurrences. Let n be the number of distinct items in S. Then, a standard Bloom Filter with a bitmap of m bits, k independent hash functions, has a error rate

$$E_r = (1 - e^{-kn/m})^k \quad (5)$$

which is minimized when $k = \ln(2) \cdot (n/m)$.

Claim 1 *For any item $e \in S$, estimate(f_e) $\geq f_e$. That is, the time-decayed frequency counts estimated by TBF never undercount real time-decayed frequency counts.*

Proof: Since each counter was set at least by one item, the at any time instance, the accumulated count values are no less than real frequency counts. TBF and Equation (1) are decaying at the same speed and at the same time instance (the same T), thus *estimate*(f_e)$\geq f_e$. □

Now we analyze the improvement of space efficiency under a Zipfian distribution of factor z. Let the multiplicity n_i of the ith most frequent to be $n_i = C/i^z$. Suppose there are n distinct items in the stream and $N > n$ is the length of the stream. Given the following condition,

$$H_n(z) = \sum_{i=1}^{n} 1/i^z, \quad N = \sum_{i=1}^{n} n_i \Rightarrow C = N / H_n(z),$$ the space improvement can be derived as follows,

$$n \log N - \sum_{i=1}^{n} \log n_i = n \log H_n(z) + z \log n!$$

Note that $H_n(z)$ is decreasing with z, $H_n(0)=n$, $H_n(1)=H_n$ (harmonic number, $H_n = \ln(n)+\Theta(1)$)

5. Experiment Evaluation

We did experiment using real web proxy log data whose characteristics are depicted in **Fig.1**. The x-axes of the plots are number of items; the y-axes are occurrence frequencies for the corresponding items. The data set exhibits strong bias.

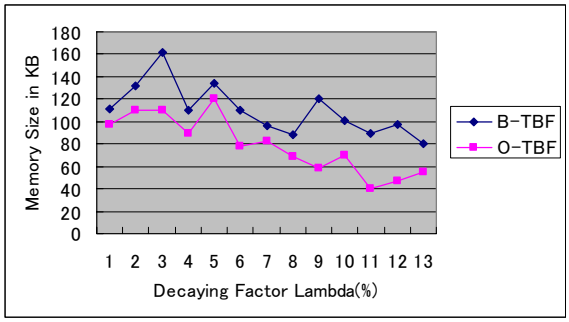

Fig. 3 Memory Usage for Basic TBF (B-TBF) and Optimized TBF (O-TBF) under Different Decaying Factors

We evaluated time and space complexity for basic TBF with 16 bit counters, and optimized TBF with 4 bit counters. The parameters for both basic TBF and the TBF1 in the optimized TBF are the same. The results are shown in **Fig. 3**. According to this figure, we can see that the optimized TBF outperforms basic in space by a factor of 2 in terms of space usage. The creation time of basic TBF is 10% faster than optimized TBF.

6. Concluding Remarks

In this paper, we have presented succinct data structures and algorithms for efficient summarizing the hit frequency profiles of the web. We focused on two aspects. First, as the recent accesses of a web page is more important, we have to deal with time-sensitivity when maintaining usage profiles for it. Second, web usage exhibits strong skew with a small fraction of web sites accounting for most traffic. Thus we do not need allocate larger counters for all items; instead we allocate small counters for the first round. For a few "hot" items, extra counters are allocated and linked to the origin ones. We develop a novel data structure, called Time-decaying Bloom Filters, TBF, to deal with the above issues. TBFs are built on Bloom Filters. A commonly-used time-decaying scheme, exponential decay [6] is used.

[References]

[1] Bloom, B.H.: Space/Time Tradeoffs in Hash Coding with Allowable Errors. CACM 13 (1970):422-426

[2] Fang, M., Shivakumar, N., Garcia-Molina, H., Motwani, R., Ullman, J.D.: Computing Iceberg Queries Efficiently. In Proceedings of 1998 VLDB Conference pp.299-310

[3] Fan, L., Cao, P., Almeida, J., Broder, A.Z.: Summary Cache: a Scalable Wide-Area Web Cache Sharing Protocol. SIGCOMM 1998 pp. 254-265

[4] Cohen, S., Matias, Y.: Spectral Bloom Filters, In Proceedings of 2003 ACM SIGMOD Conference. 241-252

[5] Manjhi, A., Shkapenyuk, V., Dhamdhere, K., Olston, C.: Finding (Recently) Frequent Items in Distributed Data Streams. ICDE 2005, pp.767-778

[6] Cohen, E., Strauss, M.: Maintaining Time-decaying Stream Aggregates. PODS 2003 pp. 223-233

[7] Kambayashi, Y., Cheng, K.: Capacity Bound-Free Web Warehouse. CIDR 2003 pp.47-57

Kai CHENG

Dr. Cheng is an Associate Professor at Faculty of Information Science, Kyushu Sangyo University. He received his PhD degree from Graduate School of Informatics, Kyoto University in 2002. His research interests include semi-persistent data management, web and stream data mining.

Mizuho IWAIHARA

Dr. Iwaihara is an Associate Professor at Graduate School of Informatics, Kyoto University. He received his PhD degrees from Graduate School of Information Engineering Kyushu University in 1993.

Limin XIANG

Dr. Xiang is an Associate Professor at Faculty of Information Science, Kyushu Sangyo University. He received his PhD degree in computer science from Kyushu University in 1999. His research interests include algorithm design and analysis and parallel computation.

Kazuo USHIJIMA

Dr. Ushijima is a Professor and dean of the Faculty of Information Science, Kyushu Sangyo University. He received his PhD degree in computer science from Kyushu University in 1971.