Optimizing Data Popularity Conscious Bloom Filters *

Ming Zhong†
Google, Inc.
mzhong@google.com

Pin Lu Kai Shen Joel Seiferas
Department of Computer Science
University of Rochester
{pinlu, kshen, joel}@cs.rochester.edu

ABSTRACT

Bloom filters are compact set representations that support set membership queries with small, one-sided error probabilities. Standard Bloom filters are oblivious to object popularity in sets and membership queries. However, sets and queries in many distributed applications follow known, stable, highly skewed distributions (*e.g.*, Zipf-like). This paper studies the problem of minimizing the false-positive probability of a Bloom filter by adapting the number of hashes used for each data object to its popularity in sets and membership queries. We model the problem as a constrained nonlinear integer program and propose two polynomial-time solutions with bounded approximation ratios — one is a 2-approximation algorithm with $O(N^c)$ running time ($c \geq 6$ in practice); the other is a $(2+\epsilon)$-approximation algorithm with running time $O(\frac{N^2}{\epsilon})$, $\epsilon > 0$. Here N denotes the total number of distinct data objects that appear in sets or queries. We quantitatively evaluate our proposed approach on two distributed applications (cooperative caching and full-text keyword searching) driven by real-life data traces. Compared to standard Bloom filters, our data popularity-conscious Bloom filters achieve up to 24 and 27 times false-positive probability reduction for the two applications respectively. The quantitative evaluation also validates our solution's bounded approximation ratio to the optimal.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: distributed applications.

General Terms

Algorithms, experimentation, performance.

*This work was supported in part by the U.S. National Science Foundation (NSF) grants CCR-0306473, ITR/IIS-0312925, CAREER Award CCF-0448413, CNS-0615045, CCF-0621472, and by an IBM Faculty Award.

†This work was done while the author was at the University of Rochester.

PODC'08, August 18–21, 2008, Toronto, Ontario, Canada.

Keywords

Bloom filters, optimization, dynamic programming, knapsack problems.

1. INTRODUCTION

A Bloom filter [2] is a compact, lossy set representation that supports set membership queries with a small probability of false-positives. In principle, they are useful for any distributed application that requires compact set/list representations (*e.g.*, due to limited network bandwidth for communicating data) and can tolerate occasional errors. Consequently, Bloom filters have been used widely to reduce the communication cost in distributed data-intensive systems. Examples include network content delivery [4], cooperative distributed caching [7, 25], peer-to-peer information retrieval [15, 23], distributed databases [16, 19], password checkers [17, 26], and resource routing [14, 24].

In many data-intensive applications, the popularity of data objects exhibits highly skewed distributions (*e.g.*, Zipf-like) [1]. Furthermore, due to the stability of such skewness [27], significant object popularities are typically estimable from collected system traces. However, such knowledge is not utilized in standard Bloom filters, which treat each data object equally by using the same number of hashes for each object. Intuitively, one can follow the simple principle of "long encodings for important objects" to adapt the per-object hash number. Specifically, a large number of hashes should be used to encode those objects that are frequent candidates for false-positives (popular in queries but unpopular in queried sets). For other objects, fewer hashed can be used to save space without causing significant error increase.

This paper studies the problem of minimizing the false-positive probability of Bloom filters for known object popularities in sets and membership queries. We model a restricted version of the problem as a non-linear integer programming problem, where the results (numbers of hashes for data objects) must be integers within a realistic range. The lower threshold of the range is 1, which indicates that at least one hash must be assigned to each data object. The upper threshold of the range is set to ensure limited computation cost of object insertions and membership queries. Nonlinear integer programming problems are hard in general unless problem-specific features are exploited. We discover a feature in this problem that allows us to achieve polynomial-time approximability. The feature is an object importance metric that can guide the assignment of hashes — the higher the score of an object, the more hashes should be used. Such an in-order hash assignment greatly reduces the solution search space, and it leads to a 2-approximation algorithm with $O(N^c)$ running time (typically $c \geq 6$ in practice) by brute-force search. We further transform the optimization problem into a variant of the knapsack problem — with ordered, variable-value objects — and use dynamic programming to solve it. The result is a $(2+\epsilon)$-approximation algorithm with running time $O(\frac{N^2}{\epsilon})$, for any $\epsilon > 0$.

Compared to standard Bloom filters, our proposed popularity-conscious Bloom filters achieve reduced false-positive probabilities (or reduced Bloom filter sizes when the same error probability threshold is satisfied). However, popularity-conscious Bloom filters incur additional cost in offline computation (for estimating object popularities and determining the customized hash scheme) and local space (for storing the generated hash scheme). This is typically a preferable trade-off in distributed applications where network communication overhead is the dominant performance constraint. We also provide a brief discussion on the computation and local space overhead.

The remainder of this paper discusses related work, formulates our optimization problem, and then presents our solution with relevant analytical results. We also provide quantitative evaluation results on an artificial workload and two distributed applications driven by real-life data traces.

2. RELATED WORK

A recent work by Bruck *et al.* [3] pursues the same goal as ours — to minimize the Bloom filter false-positive probability for given object popularities by customizing the number of hashes for each object. Their approach is to derive an optimal real-number solution (in which the number of hashes for each object is allowed to be an arbitrary real number) and then to round the real-numbers to nearest positive integers. Since the number of hashes in practice must be an integer and restricted to a realistic range (*e.g.*, {1, 2, $\cdots$, 10}), the rounding process may lead to significant increase in the false-positive probability. From an analytical point of view, their approach provides no bound on the ratio between their performance and the optimal performance. From a practical point of view, our trace-driven quantitative evaluation results (in Section 5) show that our proposed integer solution (with bounded approximation ratio) achieves up to 10 times false-positive probability reduction compare to the nearest rounding based solution under realistic application settings.

There are also studies on optimizing Bloom filters in terms of error probabilities and space overhead. Mitzenmacher [18] proposes *compressed Bloom filters*, which adjust Bloom filter parameters in order to optimize the compressed size of Bloom filters rather than the original Bloom filter size. His results show that compressed Bloom filters can achieve up to 30% space savings compared to standard Bloom filters. Pagh *et al.* [20] propose a new multi-set data structure as a substitute for standard Bloom filters for constant lookup time, smaller space usage, and succinct hash function encodings. Hao *et al.* [8] try to reduce the false-positive rate of Bloom filters by allowing each set element choose between multiple hash families, in order to set far fewer Bloom filter bits than standard Bloom filters. In comparison, we seek to reduce Bloom filter error rate from a different angle by exploiting the knowledge on data object popularity distributions.

Standard Bloom filters only support element insertions and membership queries. Recently, many variants have been proposed to support additional operations. *Counting Bloom filters* [7] support dynamic element deletions by replacing the bits in Bloom filters with small counters. *Attenuated Bloom filters* [24] use arrays of Bloom filter to store routing path information. *Spectral Bloom filters* [6] enhance Bloom filters in order to support frequency-based queries. *Bloomier filters* [5] generalize standard Bloom filters in a way that associative arrays can also be encoded and evaluated. *Exponentially Decaying Bloom Filters* [14] probabilistically encode routing tables in a highly compressed way that allows for efficient aggregation and propagation of routing information in unstructured peer-to-peer networks. Their results are orthogonal to our work since they do not focus on reducing the false positive probability of Bloom filters.

2.1 Background: Standard Bloom Filters

A Bloom filter [2] represents an n-object set $S = \{s_1, s_2, \cdots, s_n\}$ using an array of m bits, initially all set to 0. To insert each s_i, one computes k different hash functions on s_i, producing (up to) k hash values in $\{1, 2, \cdots, m\}$, and then sets all the bits corresponding to the k hash values to 1. Note that hash functions can be reused over different objects and hence only k different hash functions are needed in total. The membership query process is similar to the insertion process: To decide whether an object x belongs to S, compute the k hash values on x, and examine all the corresponding bits. If any of them is 0, then x definitely does not belong to S; if all are 1's, then answer that it does, although with some probability of a false-positive.

Assuming there is no dependency between set elements, here we will explain how standard Bloom filters determine the optimal number of hashes (k) used per object. If k independent, uniformly random hash functions are used to construct an m-bit Bloom filter for an n-element set, then the probability that an arbitrary bit in the Bloom filter is set to 1 is:

$$\mathcal{B} = 1 - \left(1 - \frac{1}{m}\right)^{kn}. \tag{1}$$

Consequently, the false-positive probability of this Bloom filter is the probability for k hashed bits of the queried object being all 1's, or:

$$f(k) = \mathcal{B}^k = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k = e^{k \cdot \ln\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)}. \tag{2}$$

Given m and n, $f(k)$ is minimized when $\mathcal{B} = \frac{1}{2}$ (each bit in the Bloom filter has equal probability to be 0 or 1), or (for large m's):

$$k = \frac{m}{n} \cdot \ln 2, \tag{3}$$

In this case, the false-positive probability f is minimized to:

$$(\frac{1}{2})^k \approx 0.6185^{\frac{m}{n}} \tag{4}$$

3. REAL-NUMBER PROBLEM

In this section, we study the optimization of data popularity-conscious Bloom filters when the number of hashes for each object can be an arbitrary real-number. In the next section, we study the practical case in which the number of hashes must be an integer within a realistic range.

3.1 Problem Formulation

Let $V = \{x_1, x_2, \cdots, x_N\}$ denote the universe of data objects. Let m be the size of a Bloom filter and n be the size of the set S represented by the Bloom filter. Similar to the analysis for standard Bloom filters, we assume that there is no dependency between different elements of a set. Let p be the membership popularity distribution — $p(i) = \frac{Pr(x_i \in S)}{n}$. The scaling factor $\frac{1}{n}$ ensures that $\sum_{i=1}^{N} p(i) = 1.0$. Let q be the query popularity distribution — $q(i)$ is the probability that x_i gets queried. For the purpose of examining the false-positive rate of Bloom filters, the *non-member* query popularity distribution q' is more interesting than the complete query distribution, q. More specifically, $q'(i)$ is the probability that x_i occurs in a missed query (hence should return *false*). In practice, p, q' are often stable and hence can be easily estimated from set and membership query traces collected during application execution [1, 27].

Let k_i be the number of hashes used for x_i. The false-positive probability of a Bloom filter, the probability that a non-member query receives a positive answer is:

$$\begin{aligned}&\sum_{i=1}^{N} q'(i) \cdot Pr(\text{ the } k_i \text{ bits that } x_i \text{ is hashed to}\\ &\text{are all 1's in the Bloom filter }) \quad = \quad \sum_{i=1}^{N} q'(i) \cdot \mathcal{B}^{k_i},\end{aligned} \tag{5}$$

where $\mathcal{B}$, the probability that an arbitrary bit of the Bloom filter is 1, is $1 - \left(1 - \frac{1}{m}\right)^{\sum_{x_i \in S} k_i}$. In other words, we want to choose k_i's in a way that the false-positive probability is minimized, while ensuring that the total number of hash operations for set S satisfies $\sum_{x_i \in S} k_i = \frac{\ln(1-\mathcal{B})}{\ln(1-\frac{1}{m})}$. In practice, there are many sets, each of which may lead to different solutions of k_i's. Given that, here we only require that the expected value for $\sum_{x_i \in S} k_i$ (over all set S's of size n) is equal to $\frac{\ln(1-\mathcal{B})}{\ln(1-\frac{1}{m})}$, *i.e.*, $\sum_{i=1}^{N} Pr(x_i \in S) \cdot k_i = \sum_{i=1}^{N} n \cdot p(i) \cdot k_i = \frac{\ln(1-\mathcal{B})}{\ln(1-\frac{1}{m})}$. Note that the actual value of a specific set may deviate from the expected value. When k_i's are restricted to certain ranges later in Section 4, such deviation can be shown to be asymptotically small for large n's with high probability (according to Hoeffding's inequality [9]).

Therefore, we define our optimization problem as finding $k_1, k_2, \cdots, k_N$ that:

$$\begin{aligned}&\text{Minimize} \quad \sum_{i=1}^{N} q'(i) \cdot \mathcal{B}^{k_i},\\ &\text{subject to} \quad \sum_{i=1}^{N} p(i) \cdot k_i = \frac{\ln(1-\mathcal{B})}{n \cdot \ln(1-\frac{1}{m})}.\end{aligned} \tag{6}$$

Let $\mathcal{K} = \frac{\ln(1-\mathcal{B})}{n \cdot \ln(1-\frac{1}{m})}$. Then the constraint in (6) can be rewritten as $\sum_{i=1}^{N} p(i) \cdot k_i = \mathcal{K}$.

3.2 Optimal Solution to the Real-Number Problem

We use the Lagrange multiplier to derive the optimal solution to (6). As there is just a single constraint, we use only one multiplier λ to combine the constraint and the optimization goal together into the Lagrangian function:

$$\Lambda(k_1, k_2, \cdots, k_N, \lambda) = \sum_{i=1}^{N} q'(i) \cdot \mathcal{B}^{k_i} - \lambda \cdot (\sum_{i=1}^{N} p(i) \cdot k_i - \mathcal{K}). \tag{7}$$

The critical values of Λ is achieved only when its gradients on $k_1, k_2, \cdots, k_N, \lambda$ are all zero. Concerning its gradients on k_i's, we have:

$$\begin{aligned}\frac{\partial \Lambda}{\partial k_i} &= q'(i) \cdot \ln \mathcal{B} \cdot \mathcal{B}^{k_i} - \lambda \cdot p(i) = 0 \Longrightarrow\\ \mathcal{B}^{k_i} &= \frac{1}{\ln \mathcal{B}} \cdot \lambda \cdot \frac{p(i)}{q'(i)}, \qquad 1 \le i \le N.\end{aligned} \tag{8}$$

Therefore, we know that the optimal solution of $k_1, k_2, \cdots, k_N$ to the real-number problem must satisfy:

$$\mathcal{B}^{k_i} \propto \frac{p(i)}{q'(i)} \Longrightarrow k_i = C + \log_{\frac{1}{\mathcal{B}}} \frac{q'(i)}{p(i)}, \qquad 1 \le i \le N, \tag{9}$$

where C is a constant independent of i.

Concerning the Lagrangian function Λ's gradient on λ, we have:

$$\frac{\partial \Lambda}{\partial \lambda} = 0 \Longrightarrow \sum_{i=1}^{N} p(i) \cdot k_i - \mathcal{K} = 0. \tag{10}$$

Thus we can compute C in (9) as follows:

$$\begin{aligned}&\sum_{i=1}^{N} p(i) \cdot k_i = \mathcal{K}\\ \Longrightarrow &\sum_{i=1}^{N} \left[p(i) \cdot \left(C + \log_{\frac{1}{\mathcal{B}}} \frac{q'(i)}{p(i)} \right) \right] = \mathcal{K}\\ \Longrightarrow &C + \sum_{i=1}^{N} p(i) \cdot \log_{\mathcal{B}} \frac{p(i)}{q'(i)} = \mathcal{K}\\ \Longrightarrow &C = \mathcal{K} + D_{p||q'} \cdot \log_{\mathcal{B}} \frac{1}{2},\end{aligned} \tag{11}$$

where $D_{p||q'} = \sum_{i=1}^{N} p(i) \cdot \log_2 \frac{p(i)}{q'(i)}$ is the *Kullback-Leibler divergence* [13] (or information divergence) between p, q', which naturally measures the deviation of q' from p. Putting together (9) and (11), we have the real-number solution of k_i's as:

$$k_i = \mathcal{K} + D_{p||q'} \cdot \log_{\mathcal{B}} \frac{1}{2} + \log_{\frac{1}{\mathcal{B}}} \frac{q'(i)}{p(i)}, \qquad 1 \le i \le N. \tag{12}$$

The optimization objective achieved by this real-number solution is:

$$\begin{aligned}\sum_{i=1}^{N} q'(i) \cdot \mathcal{B}^{k_i} &= \sum_{i=1}^{N} p(i) \cdot \mathcal{B}^{\mathcal{K}} \cdot \left(\frac{1}{2}\right)^{D_{p||q'}} = \mathcal{B}^{\mathcal{K}} \cdot \left(\frac{1}{2}\right)^{D_{p||q'}}\\ &= \mathcal{B}^{\frac{\ln(1-\mathcal{B})}{n \cdot \ln(1-\frac{1}{m})}} \cdot \left(\frac{1}{2}\right)^{D_{p||q'}}.\end{aligned} \tag{13}$$

Given m and n, the above optimization objective is minimized when $\mathcal{B}^{\ln(1-\mathcal{B})}$ is minimized. Therefore we have $\mathcal{B} = \frac{1}{2}$ and $\mathcal{K} = \frac{m}{n} \cdot \ln 2$ (for large m's). Consequently the solution of k_i's in (12) can be rewritten as:

$$\boxed{k_i = \frac{m}{n} \cdot \ln 2 + D_{p||q'} + \log_2 \frac{q'(i)}{p(i)}, \qquad 1 \le i \le N \quad (14)}$$

Similarly, the achieved false-positive probability in (13) can be rewritten as:

$$\boxed{\left(\frac{1}{2}\right)^{\frac{m}{n}\cdot\ln 2}\cdot\left(\frac{1}{2}\right)^{D_{p||q'}} \qquad (15)}$$

Note that in standard Bloom filters, $\frac{m}{n}\cdot\ln 2$ is the number of hashes used by each object and $\left(\frac{1}{2}\right)^{\frac{m}{n}\cdot\ln 2}$ is the achieved false-positive probability. The solution here always leads to an error probability no larger than standard Bloom filters in that $D_{p||q'} \geq 0$ is known to hold for all possible p, q''s and $D_{p||q'} = 0$ iff $p = q'$. This also suggests that standard Bloom filters minimize the false-positive probability only when $p = q'$ — their set elements and non-member queries follow the same distribution. Uniform distributions of p and q' are one special case of this condition.

More specifically, the solution of k_i consists of three parts: 1) $\frac{m}{n}\cdot\ln 2$, the number of hashes used by standard Bloom filters; 2) $D_{p||q'}$, a data distribution dependent constant that determines false-positive probability reduction; 3) $\log_2\frac{q'(i)}{p(i)}$, an individual deviation that only depends on the properties of object x_i. This result is consistent with our intuition that an object appearing very often in missed queries but very rare in sets should use a large number of hashes. On the contrary, those objects that are popular in sets but unpopular in queries should use few hashes. In the extreme case, if object x_i only occurs in queries but never in sets, then its number of hashes should be $k_i = \infty$. Similarly, objects that only occur in sets but never in queries lead to $k_i = -\infty$.

4. RANGED INTEGER PROBLEM

In practice, the per-object hash numbers (k_i's) must be positive integers. Further, the hash number in Bloom filters directly affects the overhead for element insertion and membership queries. In order to limit such overhead, it is desirable to set an upper threshold for k_i's. We use $k_{\max}$ to denote this upper threshold.

To derive a realistic integer solution, we can round the real-number solution of k_i's (derived in Section 3.2) into the range of positive integers $[1, k_{\max}]$. Such simple rounding, which has been used in [3], may lead to unbounded degradation on the optimization target. In this section, we formulate our ranged integer problem and then present our solutions with bounded approximation ratios.

4.1 Problem Formulation

We inherit all notations in Section 3.1.

Both standard Bloom filters and the real-number problem of data popularity-conscious Bloom filters (Section 3.2) suggest that the optimal hash number assignment is reached when $\mathcal{B} = \frac{1}{2}$. Intuitively, this is also when there is an equal probability for an arbitrary bit of the Bloom filter to be 0 or 1, with which the maximum information entropy is achieved. We redefine the real-number problem (6) with the restriction of $\mathcal{B} = \frac{1}{2}$, which effectively limits the expected total number of hashes to represent a set. With this restriction, we focus on deriving an optimal strategy to distribute the same total number of hashes among individual set elements in order to minimize the false positive probability:

$$\begin{aligned} \text{Minimize } & \sum_{i=1}^{N} q'(i)\cdot\left(\frac{1}{2}\right)^{k_i}, \\ \text{subject to } & \sum_{i=1}^{N} p(i)\cdot k_i \leq \frac{m}{n}\cdot\ln 2. \end{aligned} \tag{16}$$

Note that replacing the "=" with "≤" in the constraint does not affect the optimality of the solution since larger k_i's tend to decrease the minimization target.

We add the range constraint to problem (16) to produce a *ranged real-number problem*:

$$\begin{aligned} \text{Minimize } & \sum_{i=1}^{N} q'(i)\cdot\left(\frac{1}{2}\right)^{k_i}, \\ \text{subject to } & \sum_{i=1}^{N} p(i)\cdot k_i \leq \frac{m}{n}\cdot\ln 2, \quad 1 \leq k_i \leq k_{\max}. \end{aligned} \tag{17}$$

We further formulate a *ranged integer problem*:

$$\begin{aligned} \text{Minimize } & \sum_{i=1}^{N} q'(i)\cdot\left(\frac{1}{2}\right)^{k_i}, \\ \text{subject to } & \sum_{i=1}^{N} p(i)\cdot k_i \leq \frac{m}{n}\cdot\ln 2, \quad k_i \in \{1, 2, \cdots, k_{\max}\}. \end{aligned} \tag{18}$$

Our formulation results in a nonlinear integer programming problem, for which no general polynomial-time solution is known unless problem-specific features are exploited. Below we first introduce an important metric for our problem — *per-object importance score* — that indicates whether more or fewer hashes should be used for each object. Using this metric, we propose a 2-approximation solution with $O(N^c)$ running time ($c \geq 6$ in practice) and a $(2+\epsilon)$-approximation algorithm with running time $O(\frac{N^2}{\epsilon})$, $\epsilon > 0$.

4.2 Per-Object Importance Scores

Although the real-number solution in (14) is not directly applicable to configuring Bloom filters in practice, it motivates us to use $\frac{q'(i)}{p(i)}$ as the importance score for data object x_i — the higher $\frac{q'(i)}{p(i)}$ is, the more hashes should be used for x_i. Such scores are consistent with our intuition that an object appearing very often in missed queries but very rare in sets should use a large number of hashes. On the other hand, those objects that are popular in sets but unpopular in queries should use few hashes. The applicability of the importance scores in our integer programming problem is derived below in two steps. First, for the ranged real-number problem (17), Lemma 1 shows that the optimal solution for the number of per-object hashes must follow the order of importance scores. Furthermore, for the ranged integer problem (18), Lemma 2 shows that the best solution that follows the order of importance scores achieves a false-positive probability at most 2 times that of the optimal solution.

LEMMA 1. *Let $k_1, k_2, \cdots, k_N$ be an optimal solution to the ranged real-number problem (17). Then they must satisfy* $\forall i, j$, $\frac{q'(i)}{p(i)} > \frac{q'(j)}{p(j)} \Longrightarrow k_i \geq k_j$.

Proof: By contradiction. Assuming $\frac{q'(i)}{p(i)} > \frac{q'(j)}{p(j)}$ and $k_i < k_j$, there are three cases:

Case 1: $p(i) = p(j)$ (hence $q'(i) > q'(j)$).

In this case, simply exchanging k_i, k_j generates a better solution with a lower false-positive probability, which contradicts the optimality of the original solution. Specifically, the exchange does not affect the constraint in (17) since $k_i \cdot p(i) + k_j \cdot p(j) = k_j \cdot p(i) + k_i \cdot p(j)$. At the same time, the exchange reduces the false-positive probability since $q'(i) \cdot \left(\frac{1}{2}\right)^{k_i} + q'(j) \cdot \left(\frac{1}{2}\right)^{k_j} > q'(i) \cdot \left(\frac{1}{2}\right)^{k_j} + q'(j) \cdot \left(\frac{1}{2}\right)^{k_i}$.

Case 2: $p(i) > p(j)$.

We devise a new solution that achieves a lower false-positive probability. In the new solution, the number of hashes for objects x_i, x_j are changed to: $k'_i = \frac{p(j)}{p(i)} \cdot k_j + \left(1 - \frac{p(j)}{p(i)}\right) \cdot k_i$, and $k'_j = k_i$. We know $k_i \leq k'_i, k'_j \leq k_j$ and thus the new solution still satisfies the range constraint $[1, k_{\max}]$. Additionally, the new solution also maintains the other constraint in (17) because: $k'_i \cdot p(i) + k'_j \cdot p(j) = k_i \cdot p(i) + k_j \cdot p(j)$.

Finally, the false-positive probability related to objects x_i, x_j for the new solution is:

$$\begin{aligned} & q'(i) \cdot \left(\frac{1}{2}\right)^{k'_i} + q'(j) \cdot \left(\frac{1}{2}\right)^{k'_j} = \\ & q'(i) \cdot \left(\frac{1}{2}\right)^{\frac{p(j)}{p(i)} \cdot k_j + \left(1 - \frac{p(j)}{p(i)}\right) \cdot k_i} + q'(j) \cdot \left(\frac{1}{2}\right)^{k_i} \end{aligned} \tag{19}$$

Here $f(x) = \left(\frac{1}{2}\right)^x$ is strictly convex, i.e., for all $x, y, 0 < \lambda < 1$: $f(\lambda x + (1-\lambda)y) < \lambda f(x) + (1-\lambda) f(y)$. Hence we have

$$\begin{aligned} & q'(i) \cdot \left(\frac{1}{2}\right)^{k'_i} + q'(j) \cdot \left(\frac{1}{2}\right)^{k'_j} \\ <& q'(i) \cdot \left[\frac{p(j)}{p(i)} \cdot \left(\frac{1}{2}\right)^{k_j} + \left(1 - \frac{p(j)}{p(i)}\right) \cdot \left(\frac{1}{2}\right)^{k_i}\right] + q'(j) \cdot \left(\frac{1}{2}\right)^{k_i} \\ =& q'(i) \cdot \frac{p(j)}{p(i)} \cdot \left[\left(\frac{1}{2}\right)^{k_j} - \left(\frac{1}{2}\right)^{k_i}\right] + \left(q'(i) + q'(j)\right) \left(\frac{1}{2}\right)^{k_i} \\ \leq& q'(j) \cdot \left[\left(\frac{1}{2}\right)^{k_j} - \left(\frac{1}{2}\right)^{k_i}\right] + \left(q'(i) + q'(j)\right) \cdot \left(\frac{1}{2}\right)^{k_i} \\ =& q'(j) \cdot \left(\frac{1}{2}\right)^{k_j} + q'(i) \cdot \left(\frac{1}{2}\right)^{k_i} \end{aligned} \tag{20}$$

Case 3: $p(i) < p(j)$.

Similar to Case 2, we devise a new solution that achieves a lower false-positive probability. In the new solution, the number of hashes for objects x_i, x_j are changed to: $k'_i = k_j$, and $k'_j = \frac{p(i)}{p(j)} \cdot k_i + \left(1 - \frac{p(i)}{p(j)}\right) \cdot k_j$. We know $k_i \leq k'_i, k'_j \leq k_j$ and thus the new solution still satisfies the range constraint $[1, k_{\max}]$. It is easy to see that the other constraint in (17) also holds here.

Similar to (20), the false-positive probability related to objects x_i, x_j for the new solution is:

$$\begin{aligned} & q'(i) \cdot \left(\frac{1}{2}\right)^{k'_i} + q'(j) \cdot \left(\frac{1}{2}\right)^{k'_j} \\ =& q'(i) \cdot \left(\frac{1}{2}\right)^{k_j} + q'(j) \cdot \left(\frac{1}{2}\right)^{\frac{p(i)}{p(j)} \cdot k_i + \left(1 - \frac{p(i)}{p(j)}\right) \cdot k_j} \\ <& q'(i) \cdot \left(\frac{1}{2}\right)^{k_j} + q'(j) \cdot \left[\frac{p(i)}{p(j)} \cdot \left(\frac{1}{2}\right)^{k_i} + \left(1 - \frac{p(i)}{p(j)}\right) \cdot \left(\frac{1}{2}\right)^{k_j}\right] \\ =& q'(j) \cdot \frac{p(i)}{p(j)} \cdot \left[\left(\frac{1}{2}\right)^{k_i} - \left(\frac{1}{2}\right)^{k_j}\right] + \left(q'(i) + q'(j)\right) \left(\frac{1}{2}\right)^{k_j} \\ \leq& q'(i) \cdot \left[\left(\frac{1}{2}\right)^{k_i} - \left(\frac{1}{2}\right)^{k_j}\right] + \left(q'(i) + q'(j)\right) \cdot \left(\frac{1}{2}\right)^{k_j} \\ =& q'(i) \cdot \left(\frac{1}{2}\right)^{k_i} + q'(j) \cdot \left(\frac{1}{2}\right)^{k_j} \end{aligned} \tag{21}$$

■

LEMMA 2. *Let $k_1, k_2, \cdots, k_N$ be an optimal solution to the ranged real-number problem (17). Then $\lfloor k_1 \rfloor, \lfloor k_2 \rfloor, \cdots, \lfloor k_N \rfloor$ is a 2-approximation solution to the ranged integer problem (18) and it also satisfies $\frac{q'(i)}{p(i)} > \frac{q'(j)}{p(j)} \implies \lfloor k_i \rfloor \geq \lfloor k_j \rfloor$.*

Proof: Obviously $\lfloor k_1 \rfloor, \lfloor k_2 \rfloor, \cdots, \lfloor k_N \rfloor$ fall into $[1, k_{\max}]$. In addition, $\frac{q'(i)}{p(i)} > \frac{q'(j)}{p(j)} \implies \lfloor k_i \rfloor \geq \lfloor k_j \rfloor$ holds since $k_i \geq k_j \implies \lfloor k_i \rfloor \geq \lfloor k_j \rfloor$. We show below that this rounding process at most doubles the original optimal false-positive probability.

Let $\mathcal{B}, \mathcal{B}'$ represent the probability that an arbitrary bit of the Bloom filter is 1 when k_i's, $\lfloor k_i \rfloor$'s are used, respectively. It is obvious that $\mathcal{B}' \leq \mathcal{B} = \frac{1}{2}$ since $\lfloor k_i \rfloor \leq k_i$ — fewer hashes are used after the rounding. Furthermore, $\lfloor k_i \rfloor > k_i - 1$ always holds. Hence the false-positive probability of the rounded solution is:

$$\sum_{i=1}^{N} q'(i) \cdot (\mathcal{B}')^{\lfloor k_i \rfloor} \leq \sum_{i=1}^{N} q'(i) \cdot \mathcal{B}^{k_i - 1} = 2 \cdot \sum_{i=1}^{N} q'(i) \cdot \mathcal{B}^{k_i},$$

which is twice the false-positive probability of the optimal ranged real-number solution, and hence is at most twice that of the optimal ranged integer solution. ■

4.3 Polynomial-Time 2-Approximation

Based on Lemma 2, we restrict the search space of our solution to those that assign hash numbers to data objects in the order of their importance scores — given data objects $x_1, x_2, \cdots, x_N$ sorted in order of their importance scores (x_i precedes x_j only if $\frac{q'(i)}{p(i)} \leq \frac{q'(j)}{p(j)}$), an object with a higher score is always assigned at least as large a number of hashes than an object with a lower score. Such a monotonic integer solution are within an approximation factor of two.

Our problem is to determine $k_{\max} - 1$ indices that partition ordered data objects into $k_{\max}$ groups and use i hashes for objects in the ith group ($i \in \{1, 2, \cdots, k_{\max}\}$). More specifically, we find integers $0 \leq l_1 \leq l_2 \leq \cdots \leq l_{k_{\max}-1} \leq N$ where l_i represents the order of the highest ordered object with i or fewer assigned hashes (we know $l_0 = 0$ and

$l_{k_{\max}} = N$). In this case, our new optimization goal and constraint become:

$$\begin{aligned} &\text{Minimize} \sum_{i=1}^{k_{\max}} \left[\left(\frac{1}{2}\right)^i \cdot \sum_{j=l_{i-1}+1}^{l_i} q'(j) \right], \\ &\text{subject to} \sum_{i=1}^{k_{\max}} \left[i \cdot \sum_{j=l_{i-1}+1}^{l_i} p(j) \right] \leq \frac{m}{n} \cdot \ln 2. \end{aligned} \tag{22}$$

THEOREM 1. *The problem in (22) is polynomial-time solvable and its optimal solution guarantees an approximation ratio of 2 for the Bloom filter optimization problem (18).*

Proof: Compared to the original problem described in (18), our new problem has a greatly reduced search space. A brute-force solution can enumerate all possibilities for $l_1, l_2, \cdots, l_{k_{\max}-1}$. Since there are less than $(N+1)^{k_{\max}-1}$ such possibilities and it takes $O(N)$ time to check the constraint and calculate the minimization target for each possible case, the brute-force solution has a polynomial-time complexity of $O(N^{k_{\max}})$ (when $k_{\max}$ is a constant). In comparison, the brute-force solution for the original ranged integer problem described in (18) requires $O(k_{\max}^N \cdot N)$ time.

From Lemma 2, it follows that the optimal solution to (22) achieves an approximation ratio of 2. ∎

4.4 Faster $(2+\epsilon)$-Approximation

Although brute-force solution of (22) does give a polynomial-time approximation algorithm, its $O(N^{k_{\max}})$ running time may not be practical. First, the constant $k_{\max}$ is unlikely to be smaller than 6, since typically $\frac{m}{n} \cdot \ln 2 \geq 3$ and $k_{\max} \geq 2 \cdot \frac{m}{n} \cdot \ln 2$. Further, N can be quite large and even in the millions (*e.g.*, the number of popular URLs in the distributed caching application and the number of search keywords in the distributed search application). Therefore we propose another algorithm that achieves an approximation ratio of $2+\epsilon$ within $O(\frac{N^2}{\epsilon})$ time for any $\epsilon > 0$ of our choice.

Let $P_0, Q_0, P_1, Q_1, \cdots, P_N, Q_N$ denote the reverse cumulative sums of p, q' for the importance-ordered sequence of data objects: $P_i = \sum_{j=i+1}^N p(j)$, $Q_i = \sum_{j=i+1}^N q'(j)$. It is obvious that $P_0 = Q_0 = 1$ and $P_N = Q_N = 0$. Then the lefthand-side of the constraint in (22) can be rewritten as follows:

$$\begin{aligned} &\sum_{i=1}^{k_{\max}} \left[i \cdot \sum_{j=l_{i-1}+1}^{l_i} p(j) \right] \\ =&k_{\max} \cdot P_{l_{k_{\max}-1}} + \sum_{i=1}^{k_{\max}-1} \left[i \cdot (P_{l_{i-1}} - P_{l_i}) \right] \\ =&1 + \sum_{i=1}^{k_{\max}-1} P_{l_i} \end{aligned} \tag{23}$$

And the minimization goal in (22) can be rewritten as

$$\begin{aligned} &\sum_{i=1}^{k_{\max}} \left[\left(\frac{1}{2}\right)^i \cdot \sum_{j=l_{i-1}+1}^{l_i} q'(j) \right] \\ =&\left(\frac{1}{2}\right)^{k_{\max}} \cdot Q_{l_{k_{\max}-1}} + \sum_{i=1}^{k_{\max}-1} \left[\left(\frac{1}{2}\right)^i \cdot (Q_{l_{i-1}} - Q_{l_i}) \right] \\ =&\left(\frac{1}{2}\right) - \sum_{i=1}^{k_{\max}-1} \left[\left(\frac{1}{2}\right)^{i+1} \cdot Q_{l_i} \right] \end{aligned} \tag{24}$$

Based on (23) and (24), the optimization problem (22) is equivalent to finding integers $0 \leq l_1 \leq l_2 \leq \cdots \leq l_{k_{\max}-1} \leq N$ that (for given $P_0, Q_0, P_1, Q_1, \cdots, P_N, Q_N$):

$$\begin{aligned} &\text{Maximize} \sum_{i=1}^{k_{\max}-1} \left[\left(\frac{1}{2}\right)^{i+1} \cdot Q_{l_i} \right], \\ &\text{subject to} \sum_{i=1}^{k_{\max}-1} P_{l_i} \leq \frac{m}{n} \cdot \ln 2 - 1. \end{aligned} \tag{25}$$

This problem can be viewed as a knapsack problem in that it selects $k_{\max}-1$ objects with maximal total value (the weighted sum of Q's) while keeping the total size of selected objects (the sum of P's) below the knapsack capacity ($\frac{m}{n} \cdot \ln 2 - 1$). However, the problem differs from standard knapsack problems in two ways: 1) A fixed number of objects are chosen. 2) Objects have variable values dependent on the order by which they are selected, *e.g.*, the first selected object has value $(\frac{1}{2})^2 \cdot Q_{l_1}$; the second selected object leads to value $(\frac{1}{2})^3 \cdot Q_{l_2}$, *etc.* Our studied variant of the knapsack problem is reminiscent of the partially ordered knapsack (POK) problem [10, 12], which requires that certain objects must be chosen only if others have been selected before. Although POK is known to be strongly NP-hard even when the partial order is bipartite [10], our variant has a fully-polynomial-time approximation scheme because its objects follow a total precedence order. Dynamic programming, a useful technique for solving standard knapsack problems, is still applicable for our variant (but with some adjustments).

For our dynamic programming algorithm, we use $f(x) = \lfloor \frac{x}{u} \rfloor \cdot u$ to round object values, where $u = \frac{\epsilon}{2} \cdot \frac{1}{k_{\max}-1} \cdot (\frac{1}{2})^{k_{\max}}$. After the rounding, each object value is a multiple of u. Let $L_{i,j,w}$ denote an i-element partial solution $l_1 \leq l_2 \leq \cdots \leq l_i \leq j$ whose total value is w and total size is minimized, where $i \in \{1, 2, \cdots, k_{\max}-1\}$, $j \in \{1, 2, \cdots, N\}$. We know $w \in \{0 \cdot u, 1 \cdot u, \cdots, \lfloor \frac{2^{-1}}{u} \rfloor \cdot u\}$ because the maximum total value (the maximization goal in (25)) never exceeds 2^{-1} and the unit value is u. Let $S_{i,j,w}$ denote the total size of $L_{i,j,w}$ and $S_{i,j,w} = \infty$ if $L_{i,j,w}$ does not exist. It is easy to directly compute $S_{1,j,w}, L_{1,j,w}$'s by a simple scan over $P_0, Q_0, P_1, Q_1, \cdots, P_N, Q_N$. Other $S_{i,j,w}$'s ($i \geq 2$) can be computed by using dynamic programming, as described below ($L_{i,j,w}$'s are updated accordingly):

$$S_{i,j,w} = \min_{y \in \{1,2,\cdots,j\}} \{P_y + S_{i-1,y,w-f(2^{-(i+1)} \cdot Q_y)}\} \tag{26}$$

After the execution of the above dynamic programming, the optimal solution to (25) is achieved by choosing $L_{k_{\max}-1,N,w}$ with $w = \max\{z | S_{k_{\max}-1,N,z} \leq \frac{m}{n} \cdot \ln 2 - 1\}$.

THEOREM 2. *For $\epsilon > 0$, our dynamic programming algorithm (26) achieves an approximation ratio of $2 + \epsilon$ for the ranged integer optimization problem (18) within $O(\frac{N^2}{\epsilon})$ time.*

Proof: Given an optimal solution to (25), let OPT represent its corresponding false-positive probability. Let OPT' represent the new false-positive probability when rounded object values of the original optimal solution are used. Since the solution chooses $k_{\max} - 1$ objects and each object has its value changed by at most u after the rounding, we know: $OPT' \leq OPT + (k_{\max} - 1) \cdot u = OPT + \frac{\epsilon}{2} \cdot \left(\frac{1}{2}\right)^{k_{\max}}$.

Our dynamic programming (26) finds the optimal solution on rounded values and hence its corresponding false-positive probability OPT'' should satisfy $OPT'' \leq OPT' \leq OPT + \frac{\epsilon}{2} \cdot \left(\frac{1}{2}\right)^{k_{\max}}$. We also know that $OPT \geq \left(\frac{1}{2}\right)^{k_{\max}}$ because every object uses at most $k_{\max}$ hashes. Hence $\frac{OPT''}{OPT} \leq 1 + \frac{\epsilon}{2}$. Furthermore, we know OPT is the optimally achieved false-positive probability for the importance score-following problem (25). According to Lemma 2, OPT is at most twice the minimal false-positive probability achieved by the optimal solution to the ranged integer problem (18). Consequently, the solution of our dynamic programming achieves an approximation ratio of $2 \cdot (1 + \frac{\epsilon}{2}) = 2 + \epsilon$ to problem (18).

The dynamic programming has $k_{\max} - 1$ rounds (i's) and each round examines at most N objects (y's) on $N \cdot \frac{1}{2u}$ table entries ((j, w)'s). Consequently, the running time of our dynamic programming is at most $(k_{\max} - 1) \cdot N \cdot \frac{1}{2u} \cdot N = O(\frac{N^2}{\epsilon} \cdot k_{\max}^2 \cdot 2^{k_{\max}}) = O(\frac{N^2}{\epsilon})$, where $k_{\max}$ is typically a small integer in practice. ■

5. QUANTITATIVE EVALUATION

Using a synthetic workload and two application case studies driven by real-life data traces, we quantitatively compare four Bloom filter approaches:

- *Standard Bloom filters*, which is oblivious to data object popularity distributions.
- *Popularity-conscious Bloom filters, real-rounding solution.* This approach uses the rounding results of the optimal real-number solution of k_i's in (14). We round each k_i into its nearest integer except that $k_i < 1$ is rounded to 1 and $k_i > k_{\max}$ is rounded to $k_{\max}$.
- *Popularity-conscious Bloom filters, integer solution.* This is our dynamic programming-based $(2+\epsilon)$-approximation ranged integer solution described in Section 4.4.
- *Popularity-conscious Bloom filters, optimal+ solution.* The false-positive probability of the optimal real-number solution in (15) indicates a loose performance upper-bound for comparison. It is unlikely that any ranged integer solution can achieve a matching performance and thus we add the plus symbol after "optimal".

5.1 Evaluation on Zipf-Like Data Popularity Distributions

We first use a simple synthetic workload in which the non-member query popularity distribution q' is Zipf-like — $q'(i) \propto i^{-\alpha}, 1 \leq \alpha \leq 2$; and the membership popularity distribution p is uniform — $p(i) = \frac{1}{N}$. Under varying skewness parameter α in the Zipf distribution, Figure 1 shows the false-positive probability achieved by different Bloom filter approaches. Results suggest that the simple rounding may lead to a significant increase (2–400 times depending on α) in false-positive probability over the optimal real-number solution. Such rounding-induced error increase is more pronounced with larger difference between p, q'. In comparison, our proposed integer solution for popularity-conscious Bloom filters only leads to a moderate and stable increase (around two times or less) in the false-positive rate over all test cases.

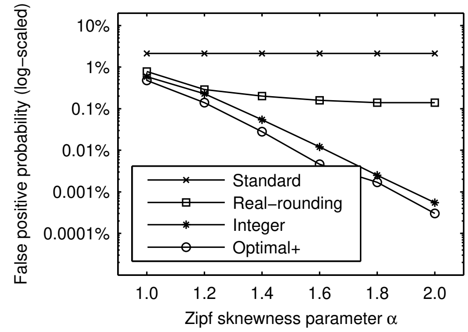

Figure 1: Comparison of Bloom filter approaches on synthetic Zipf-distributed data popularities under different Zipf skewness. In the experiments, $\frac{m}{n} = 8$ (averaging 8 Bloom filter bits per set element) and $k_{\text{standard}} = 5$ for standard Bloom filters. We set $k_{\max} = 2 \cdot k_{\text{standard}}$.

5.2 Trace-driven Case Study on Cooperative Distributed Caching

Web caching is an effective and widely used technique to reduce web traffic and Internet bandwidth consumption. The caching efficiency can be further improved by sharing cached data among distributed web caches [7,25]. In particular, Fan *et al.* proposed *summary cache* to support efficient cache sharing [7]. In their approach, each cache maintains a summary of the content at every other cache. Whenever a local miss occurs, the cache searches all the summaries for the requested data object and forwards the query to a cache whose content summary indicates that it possesses the object. Since cache summaries need to be compact and they can tolerate some false-positives, Bloom filters are employed to represent the cache summaries. In this section, we will present the performance improvement of using our data popularity-conscious Bloom filters rather than standard Bloom filters for supporting cooperative distributed caching.

Data Traces and Evaluation Setup. Our evaluation is based on data traces from National Lab of Applied Network Research (NLANR) Web Caching project (`www.ircache.net`), which consists of ten cooperative web caches. We use the traces of 3 weeks (09/15/2005–10/5/2005, 42.7 million web requests with 15.9 million distinct URLs) as training dataset to acquire set distribution p and non-member query distribution q'. Then we use the algorithms proposed in Section 4.4 to compute customized hash schemes for objects (URLs).

The performance comparison between our Bloom filters and standard Bloom filters are conducted based on the traces of next 3 weeks (10/06/2005–10/27/2005, 47.7 million web

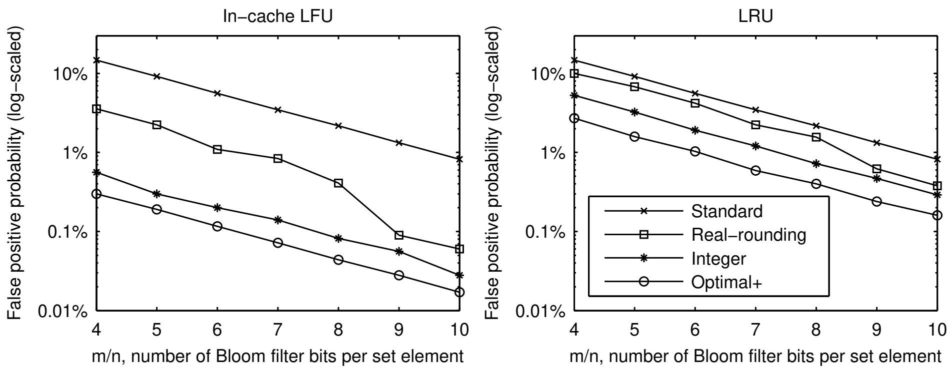

Figure 2: Comparison of Bloom filter approaches with varying Bloom filter sizes for the case study of cooperative distributed caching We show the comparison results using two cache replacement policies. In the experiments, we set $k_{\max}$ (upper threshold for per-object hash number in popularity-conscious Bloom filters) as twice the number of per-object hashes in standard Bloom filters.

requests with 17.1 million distinct URLs). In the evaluation phase, we use a discrete event simulator to simulate distributed web caching system on the testing data traces. The simulator reads cache requests from the traces at ten proxies and merges them into one queue (in the time order) based on the timestamp of each request. For the cache requests in the queue, the simulator processes them in their time order and updates their corresponding caches accordingly. As suggested by Fan *et al.*, our simulator broadcasts the content summary of a cache to all other caches whenever the cache has evicted 1% of its members since last broadcast [7]. When a request misses in a cache, the simulator checks the up-to-date received summaries at that cache to see if the requested object may be stored in other caches. If it appears promising, the simulator forwards the request to the appropriate cache and checks whether it is a false hit or not.

Throughout the training and evaluation phases, we use a cache size of 100,000 objects. We consider two common cache replacement policies in our study:

- *In-cache LFU* keeps a reference count for each cached document and replaces the least frequently visited document with the incoming document if necessary (LRU policy is used to break ties). Note that the in-cache LFU does not maintain reference history for evicted documents and therefore its reference frequency information may be incomplete. There exist a variety of other approaches that employ more complete reference frequency information, such as LFU-aging and weighted LFU [21]. However, a simple scheme like in-cache LFU is sufficient for our illustration purpose.
- *LRU* organizes cache content as a queue and always puts the most recently visited object at the tail of the queue. The object at the head of the LRU queue (the least recently requested object) will be removed at each cache replacement.

Simulation Results. We compare the four Bloom filter approaches defined earlier — standard Bloom filters, popularity-conscious real-rounding solution, popularity-conscious integer solution derived by our proposed algorithm, and popularity-conscious optimal+ solution. Figure 2 shows that our proposed popularity-conscious integer solution significantly outperforms alternative schemes. Compared to standard Bloom filters, our approach's false-positive probability reduction is at least 24 times under in-cache LFU, and around 3 times under LRU. Compared to popularity-conscious real-rounding solution, our approach's false-positive probability reduction is 1.6–7.5 times under in-cache LFU, and 1.3–2.1 times under LRU. At the same time, the false-positive probability achieved by our popularity-conscious integer solution is always within a factor of 2 from the optimal+ solution.

5.3 Trace-driven Case Study on Distributed Full-text Keyword Search

Most of the current distributed full-text keyword search systems are based on distributed keyword indices, where each node maintains the inverted lists of some keywords. A query with $k \geq 1$ keywords needs to contact all nodes that hold the inverted lists of the k keywords and computes the intersection of these inverted lists. For large datasets, the performance of distributed keyword indices is dominated by the efficiency of set intersection operations.

Given sets A, B with $|A| \leq |B|$, we compute $A \cap B$ as follows:
1. N_A sends the Bloom filter representation of A to N_B.
2. Every object in B will be queried to A's Bloom filter. N_B then sends the set $S = \{x | x \in B \wedge Answer = YES\}$ to N_A.
3. N_A computes $S \cap A$, which is equal to $A \cap B$ since $A \cap B \subseteq S \subseteq B$.

Figure 3: Set intersection supported by Bloom filters.

Li *et al.* [15] and Reynold and Vahdat [23] suggest to use Bloom filters to reduce the communication overhead of computing the set intersection between sets A, B from different nodes N_A, N_B. For $|A| \leq |B|$, $A \cap B$ can be computed by using Bloom filters as shown in Figure 3. Compared to the naive way of computing $A \cap B$ where A is directly sent over the network, the Bloom filter approach significantly reduces the communication overhead by sending the compact

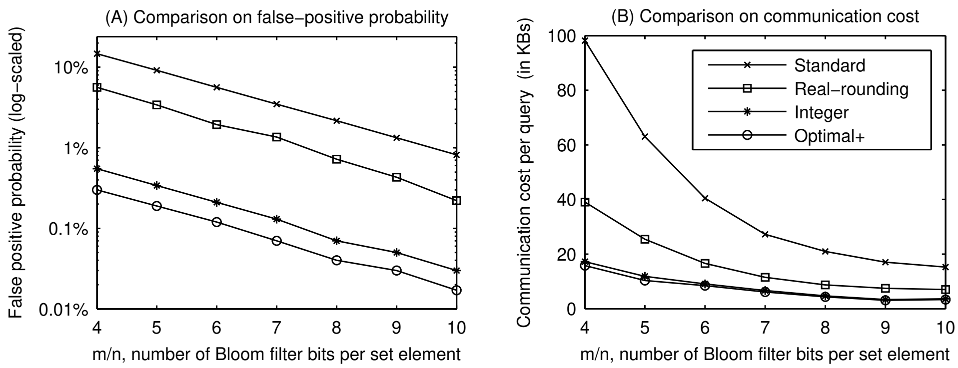

Figure 4: Comparison of Bloom filter approaches with varying Bloom filter sizes for the case study of distributed full-text keyword search. We show the comparison results on false-positive probability (A) and on communication overhead (B). In the experiments, we set $k_{\max}$ (upper threshold for per-object hash number in popularity-conscious Bloom filters) as twice the number of per-object hashes in standard Bloom filters.

representation of A over the network. Note that this approach maintains perfect accuracy by resolving the Bloom filter false-positives in step 3. However, more false-positives result in more communication overhead in step 2. In this section, we will present the communication overhead saving of using our data popularity-conscious Bloom filters rather than standard Bloom filters in set intersection operations.

Data Traces and Simulation Setup. Our dataset contains 3.7 million web pages and 6.8 million web queries. The web pages are crawled based on URL listings of the Open Directory Project (`www.dmoz.com`). The queries are from a partial query log at the Ask Jeeves search engine (`www.ask.com`) over the week of 01/06/2002–01/12/2002 and there are an average of 2.54 terms per query. There are 253,334 distinct words that appear in the query log. In our simulation, we randomly distribute the 253,334 corresponding inverted lists (one inverted list for each word) over 4096 virtual nodes, where each node is responsible for the inverted lists of some words. For each incoming query, our simulator computes the intersection of the inverted lists of its query terms as follows. For a query that consists of words $w_1, w_2, \cdots, w_k$ with ascending order of inverted list sizes, the simulator visits their hosts in the same order and computes $S_1 \cap S_2$, $(S_1 \cap S_2) \cap S_3$, ..., $(S_1 \cap S_2 \cap ...S_{k-1}) \cap S_k$ step by step, where $S_1, S_2, \cdots, S_k$ are the inverted lists for the words $w_1, w_2, \cdots, w_k$ and each intersection is computed in a way as described in Figure 3.

According to Figure 3, if the Bloom filter of A uses a total of m bits and its false-positive probability is P, then the total algorithm communication overhead is:

$$\frac{m}{8} + |A \cap B| \cdot 4 + |B - B \cap A| \cdot P \cdot 4 \ (bytes) \tag{27}$$

where $\frac{m}{8}$ $(bytes)$ is the overhead for step 1 and $|A \cap B| \cdot 4 + |B - B \cap A| \cdot P \cdot 4$ $(bytes)$ is the overhead related to step 2. Compared to standard Bloom filters, popularity-conscious Bloom filters have a lower false-positive probability (P), which leads to lower overall communication overhead when the same Bloom filters size is used.

Simulation Results. Figure 4 shows that our proposed popularity-conscious integer solution significantly outperforms alternative schemes. In Figure 4(A), our approach reduces the Bloom filter false-positive probability by around 27 times compared to standard Bloom filters, and by around 10 times compared to the popularity-conscious real-rounding solution. At the same time, the false-positive probability achieved by our popularity-conscious integer solution is always within a factor of 2 from the optimal+ solution. In Figure 4(B), our approach saves the communication overhead by 76–82% compared to standard Bloom filters, and by 49–56% compared to the popularity-conscious real-rounding solution. At the same time, our approach incurs <10% more communication overhead than the optimal+ solution does.

6. CONCLUSION AND DISCUSSION

We discover that standard Bloom filters, which are oblivious to data object popularities, are not optimal unless set distributions and non-member query distributions are identical. Motivated by that, this paper studies the problem of minimizing the false-positive probability of Bloom filters by adapting the number of hashes used for each object to its popularities in sets and membership queries. Guided by a novel object importance metric, we propose the first polynomial-time integer solution with bounded approximation ratios. Our results include a 2-approximation algorithm with $O(N^c)$ running time ($c \geq 6$ in practice) and a $(2+\epsilon)$-approximation algorithm with running time $O(\frac{N^2}{\epsilon})$, $\epsilon > 0$.

We provide quantitative evaluations using one artificial workload and two application case studies driven by real-life data traces. Evaluation results show that our proposed popularity-conscious Bloom filters can achieve significant false-positive probability reduction compared to standard Bloom filters and the simple popularity-conscious solution that rounds optimal real-number solutions into nearest integers.

This work is related to our other studies of using known object popularity information to customize object replication degrees [27] and object placement for multi-object operations [28]. Together, our results demonstrate substantial system adaptation benefits by exploiting skewed but stable data access distributions in real-world data-intensive applications.

Discussion on Overhead. Consider a practical implementation of our data popularity-conscious Bloom filters in which each data object and its number of hashes is stored in a local lookup table. In such an implementation, the lookup table size is proportional to the total number of distinct data objects. Further, Theorem 2 shows that the offline computation time is quadratic in the total object count. The total object count can be as high as millions in practice (*e.g.*, the number of keywords in web pages) and therefore the space and offline computation overhead may be too expensive. To control such overhead, one may apply *limited optimization* — to only optimize a small number of objects with highest and lowest $\frac{q'(i)}{p(i)}$. For the remaining objects with median $\frac{q'(i)}{p(i)}$ values, we will just use the default number of per-object hashes as in standard Bloom filters. We believe that the performance improvement of very limited optimization is still significant in applications with highly skewed popularities. Specifically, the most popular objects in queries typically have high $\frac{q'(i)}{p(i)}$ values while the most popular objects in sets often have low $\frac{q'(i)}{p(i)}$ values. With high skewness, a small number of most popular objects dominate the overall popularity. For example, for a 10^6-element dataset with its popularity distribution following Zipf's law ($Prob(i) \propto \frac{1}{i}$), a limited optimization to the most popular 10^3 objects accounts for half of the combine popularity.

7. REFERENCES

[1] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira. Characterizing reference locality in the WWW. In *the 4th Int'l Conf. on Parallel and Distributed Information Systems*, pages 92–103, 1996.

[2] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[3] J. Bruck, J. Gao, and A. Jiang. Weighted Bloom filters. In *the IEEE Int'l Symp. on Information Theory*, July 2006.

[4] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost. Informed content delivery across adaptive overlay networks. In *ACM SIGCOMM*, pages 47–60, 2002.

[5] B. Chazelle, J. Kilian, R. Rubinfeld, and A. Tal. The Bloomier filter: An efficient data structure for static support lookup tables. In *the 15th Symp. on Distributed Algorithms*, pages 30–39, 2004.

[6] S. Cohen and Y. Matias. Spectral Bloom filters. In *the ACM SIGMOD*, pages 241–252, 2003.

[7] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. on Networking*, 8(3):281–293, 2000.

[8] F. Hao, M. Kodialam, and T.V. Lakshman. Building high accuracy Bloom filters using partitioned hashing. In *ACM SIGMETRICS*, pages 277–288, 2007.

[9] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[10] D. S. Johnson and K. A. Niemi. On knapsack partitions and a new dynamic programming techniques for trees. *Mathematics of Operations Research*, 8(1):1–14, 1983.

[11] A. Kirsch and M. Mitzenmacher. Less hashing, same performance: Building a better Bloom filter. In *the 14th European Symp. on Algorithms*, pages 456–467, 2006.

[12] S. G. Kolliopoulos and G. Steiner. Partially ordered knapsack and applications to scheduling. In *the 10th European Symp. on Algorithms*, pages 612–624, 2002.

[13] S. Kullback and R.A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:79–86, 1951.

[14] A. Kumar, J. Xu, and E. W. Zegura. Efficient and scalable query routing for unstructured peer-to-peer networks. In *the IEEE INFOCOM*, 2005.

[15] J. Li, T. Loo, J. Hellerstein, F. Kaashoek, D.R. Karger, and R. Morris. On the feasibility of peer-to-peer web indexing and search. In *the 2nd Int'l Workshop on Peer-to-Peer Systems*, 2003.

[16] L. F. Mackert and G. M. Lohman. R* optimizer validation and performance evaluation for distributed queries. In *the 12th Int'l Conf. on Very Large Data Bases*, pages 149–159, 1986.

[17] U. Manber and S. Wu. An algorithm for approximate membership checking with application to password security. *Information Processing Letters*, 50:191–197, 1994.

[18] M. Mitzenmacher. Compressed bloom filters. In *the 20th ACM Symp. on Principles of Distributed Computing*, pages 144–150, 2001.

[19] J. K. Mullin. Optimal semijoins for distributed database systems. *IEEE Trans. on Software Engineering*, 16(5):558–560, 1990.

[20] A. Pagh, R. Pagh, and S. S. Rao. An optimal Bloom filter replacement. In *the 16th ACM Symp. on Discrete Algorithms*, pages 823–829, 2005.

[21] S. Podlipnig and L. Boszormenyi. A survey of web cache replacement strategies. *ACM Computing Surveys*, 35(4):374–398, 2003.

[22] M. V. Ramakrishna. Practical performance of Bloom filters and parallel free-text searching. *Communications of ACM*, 32(10):1237–1239, 1989.

[23] P. Reynold and A. Vahdat. Efficient peer-to-peer keyword searching. In *Middleware*, pages 21–40, 2003.

[24] S. Rhea and J. Kubiatowicz. Probabilistic location and routing. In *the 21st IEEE INFOCOM*, pages 1248–1257, 2002.

[25] Y.J. Song, V. Ramasubramanian, and E.G. Sirer. Optimal resource utilization in content distribution networks. Technical Report TR2005-2004, Dept. of Computer Science, Cornell Univ., 2005.

[26] E. H. Spafford. Opurs: Preventing weak password choices. *Computer and Security*, 11:273–278, 1992.

[27] M. Zhong, K. Shen, and J. Seiferas. Replication degree customization for high availability. In *the 3rd EuroSys Conf.*, pages 55–68, 2008.

[28] M. Zhong, K. Shen, and J. Seiferas. Correlation-aware object placement for multi-object operations. In *the 28th Int'l Conf. on Distributed Computing Systems*, 2008.