Access-Efficient Balanced Bloom Filters

Yossi Kanizo
Dept. of Computer Science
Technion, Haifa, Israel
ykanizo@cs.technion.ac.il

David Hay
School of Computer Science And Engineering
Hebrew University., Jerusalem, Israel
dhay@cs.huji.ac.il

Isaac Keslassy
Dept. of Electrical Engineering
Technion, Haifa, Israel
isaac@ee.technion.ac.il

***Abstract*—The implementation of Bloom Filters in network devices ideally has low memory access-rate and low false positive rate. However, since Bloom Filters hash elements to arbitrary memory blocks, they need high memory access rates. On the other hand, Blocked Bloom Filters first hash elements to a single memory block, where they maintain a local Bloom Filter. Therefore, they access only one memory block per element, resulting in better memory-access efficiency. Unfortunately, they have poor performance, with poor load-balancing and high false positive rates.**

In this paper, we propose to implement load-balancing schemes for the choice of the memory block, along with an optional overflow list, resulting in improved false positive rates while keeping a high memory-access efficiency. To study this problem, we define, analyze and solve a fundamental *access-constrained balancing problem*, where incoming elements need to be optimally balanced across resources while satisfying average and instantaneous constraints on the *number of memory accesses* associated with checking the current load of the resources. We then build on this problem to suggest a new load-balanced Blocked Bloom Filter scheme. Finally, we show that this scheme can reduce the false positive rate by up to two orders of magnitude, with the cost of 1.2 memory accesses per element and an overflow list size of 0.5% of the elements.

I. Introduction

Bloom Filter is a space-efficient randomized data structure that supports approximate set membership queries [1]. Its quality is measured by its false positive rate (FPR), i.e. the probability that a set membership query returns TRUE, while the element is not in the set; Bloom Filters always have zero false negative rate. Bloom Filters are often used in network applications, especially when the set is very large, when the memory is scarce (e.g. high-speed on-chip memory), or when memory should be shared across many nodes in a limited-bandwidth network [2], [3].

Unfortunately, while efficient in memory space, *Bloom Filters require significant number of memory accesses*. For instance, a Bloom Filter with 30 bits per element yields negligible FPR, but also requires to access about $30 \cdot \ln 2 \approx 21$ memory bits per element. Note that each of these bits can reside in an arbitrary location over the memory space. Thus, such a Bloom Filter would need a prohibitive memory access bandwidth when either implemented in an off-chip setting (that is, requiring accessing 21 memory blocks per element) or distributed over a network (equivalently, it may be required to access 21 nodes per element).

One proposal to improve the access-efficiency of Bloom Filters is to use a Blocked Bloom Filter [4], [5], in which each element is first hashed using a *single* hash function to one of the memory blocks, and then the memory block operates a local Bloom Filter. Although this technique is clearly access-efficient, since each element requires to access a single memory block, it also suffers from a high false positive rate, due to a typical *load imbalance* between the memory blocks.

To tackle this problem, our basic approach is to *use load-balancing schemes*, making the number of elements in each memory block as balanced as possible. We further propose to use an overflow list that stores elements hashed to overloaded memory-blocks. The overflow list is typically small, thus for example, it can be implemented using a content-addressable memory (CAM), which supports parallel lookup operation.

We study this problem using a *general* access-efficient approach to the *balancing problem*—a fundamental problem that lies at the core of many operations and applications in modern distributed and communication systems.

We model the balancing problem using the *balls and bins model* [6], and more specifically its *sequential multiple-choice* variant [7]. In this model, n balls are placed in m bins. Before placing a ball, d bins are chosen according to some distribution (e.g., uniformly at random) and the ball is placed in one of these bins following some rule (for example, in the least occupied bin). Moreover, we consider an extension of this model that allows a small fraction γ of the balls not to be placed in the bins; these balls are either disregarded or stored in a dedicated overflow list, usually implemented in an expensive memory (a similar model was considered, for example, in [8]). The quality of the balancing is measured by the load on the bins: The resulting load at each bin induces a certain *cost*, which is calculated by an arbitrary non-decreasing convex *cost function* ϕ. Our goal is to minimize the overall expected cost of the system.

We further impose the following restriction: each operation can look at up to $a < d$ bins *on average*, before deciding where to place the ball. Note that in most reasonable scenarios, checking the status of a bin (e.g., its occupancy) corresponds to either a memory access or a probe over the network. Thus, our restriction can be viewed as imposing a *memory access budget* on the insertion algorithm. *Given this access budget, we aim at achieving the highest-quality balancing.*

Our Contributions: In this paper, we propose access-efficient balanced Bloom Filters. We first maintain load-balancing schemes to distribute the elements between the memory blocks. The basic way of operation of these schemes is to query the load of the hashed memory block, and in

case the load exceeds some threshold, another hash function is used. We also propose to use an optional overflow list to store elements that all hash functions used map to already overloaded memory blocks.

To study this problem, we explore the *optimality region* of the balancing problem. Namely, we consider different balancing schemes at different loads, and determine several selections of the access budget a and the overflow fraction γ such that the balancing scheme is *optimal* with respect to the cost function ϕ. In particular, given some access budget a within a predetermined range, we will show that our scheme is optimal for some $\gamma(a)$, and for any γ satisfying $\gamma \geq \gamma(a)$, thus defining an *optimality region* over the (a, γ) plane.

To show optimality, we first provide lower bounds on the minimum cost of each instance of the problem. The lower bound depends on the access budget a, the number of hash functions d, and the overflow list size, but, quite surprisingly, does not depend on the cost function ϕ. Our lower bounds hold when all hash functions have uniform distribution or when their overall distribution is uniform (in the latter case, the hash function distributions can be different).

Then, we provide three different schemes that meet the lower bounds on different access budgets; we further find the minimum size of the overflow list that should be provided in order to achieve optimality. All our analytical models are compared with simulations showing their accuracy.

We conclude by showing how, with a proper choice of the cost function ϕ, the balancing problem can be directly used to optimize Bloom Filters. For example, for an average number of access operations of 1.2, and 0.5% of the elements stored in the overflow list, the false positive rate can be reduced by up to two orders of magnitude.

Paper Organization: We first describe our basic architecture of access-efficient Bloom Filters in Section II. Then, the optimal balancing problem is defined in Section III, followed by our lower bound results in Section IV. The three optimal schemes and their analysis are presented in Sections V, VI, and VII, while a comparative study appears in Section VIII. In Section IX we show how the solution of the balancing problem can be used to construct access-efficient Bloom Filters. Finally, Section X surveys related work.

Due to space limits, the detailed proofs and additional applications of the access-efficient load-balancing problem are presented in [9].

II. Access-Efficient Balanced Bloom Filters

In this section we present the basic architecture of the access-efficient balanced Bloom Filter. Our architecture follows the two guidelines of balancing the elements between the memory blocks and the usage of an overflow list.

In our basic architecture, each memory block functions as a local Bloom Filter, with the only modification that the memory block also saves some bits for a counter storing the number of elements inserted locally (although in practice, the actual load can be also determined quite accurately by the number of set bits).

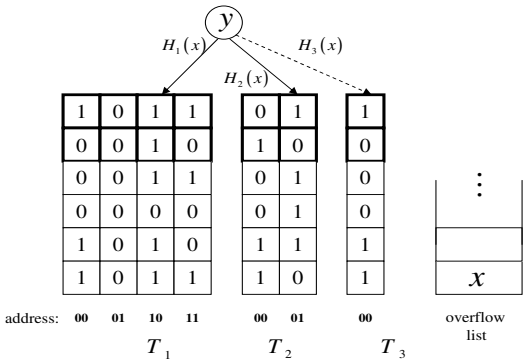


Fig. 1. Illustration of a Bloom Filter implementation using MHT, with three subtables. T_2 only holds elements that did not fit T_1; T_3 only holds elements that did not fit T_2. The upper two bits in each memory block, depicted in bold, are used to count the number of elements.

Although any balancing scheme can be used in general, for the purpose of the section we take a *special case* of the *multi-level hash table* (MHT) balancing scheme that is presented in full in Section VII. In the special case of the MHT balancing scheme, the memory is divided into d separate subtables $T_1, \ldots, T_d$, with a single uniform hash function mapping to each one of these subtables. Upon an element arrival, it is placed in the first subtable in which the corresponding mapped memory block has load lower than a pre-defined threshold h. If no such memory block exists, the element is placed in the overflow list. Note that, as described in Section VII, in the general case of the MHT balancing scheme, elements can also be stored in a bucket with load h under some conditions.

A lookup operation follows the same steps as an insertion operation: The Bloom Filters of the mapped memory blocks are queried one by one, until either the element is found or a Bloom Filter with load less than h is queried. Only if all filters have full load and the element was not found, the overflow list is also queried.

Fig. 1 illustrates an insertion of a new element into the Bloom Filter based on the special case of the MHT balancing scheme with $h = 3$. The memory consists of 3 subtables of decreasing size, with 4, 2 and 1 memory blocks, respectively. Each memory block is of size 6 bits, with 2 bits for the counter and 4 for the local bloom filter. When element y arrives, it is first hashed to the memory block at address 10 in subtable T_1. The counter at this memory block indicates that 3 elements have already been inserted there, therefore the schemes tries to insert the element into subtable T_2. In this subtable, the element is hashed to address 01, where there are 2 elements. At this point, the element is inserted to this memory block. The dashed arrow to subtable T_3 illustrates a hash function that is not actually performed.

III. Problem Statement

To further study our problem, we first define and solve the *optimal access-constrained balancing problem* in the following sections. In this section, we define the notations and settings of this balancing problem.

Let $\mathcal{B}$ be a set of m *buckets* (or *bins*) of unbounded size, and let $\mathcal{E}$ be a set of n *elements* (or *balls*) that should be distributed among the buckets. In addition, denote by $r = \frac{n}{m}$ *the element-per-bucket ratio*.

Assume also that there exists an *overflow list* [8], i.e. a special bucket of bounded size $\gamma \cdot n$ (namely, at most a fraction γ of the elements can be placed in the list), which can be used by the insertion algorithm at any time. For example, depending on the application, the overflow list may correspond to a dedicated memory—e.g., content-addressable memory (CAM)—in hardware-implemented hash-table, or to the loss ratio when the balancing scheme is allowed to drop elements.

Elements are inserted into either one of the m buckets or the overflow list, according to some balancing scheme with at most d hash-functions per element, which is defined as follows (a similar definition appears in [10]).

Definition 1: A *balancing scheme* consists of:
(i) d hash-function probability distributions over bucket set $\mathcal{B}$, used to generate a *hash-function set* $\mathcal{H} = \{H_1, \ldots, H_d\}$ of d independent random hash functions;
(ii) an *insertion algorithm* that places each element $x \in \mathcal{E}$ in one of the d buckets $\{H_1(x), \ldots, H_d(x)\}$ or in the overflow list. The insertion algorithm is an *online* algorithm, which places the elements one after the other with no knowledge of future elements.

The access-efficiency of a balancing scheme is measured by the number of *bucket accesses* needed to store the incoming elements. We assume that a balancing scheme needs to access a bucket to obtain any information on it. We do not count accesses to the overflow list.

We further consider two constraints, which can be seen as either power- or throughput-constraints depending on the application. First, we require that the *average* number of bucket accesses per element insertion must be bounded by some constant $a \geq 0$. In addition, notice that the *worst-case* number of bucket accesses per element insertion is always bounded by d, because an element does not need to consider any of its d hash functions more than once. These two constraints are captured by the following definition:

Definition 2: An $\langle a, d, r \rangle$ balancing scheme is a balancing scheme that inserts all elements with an average (respectively, maximum) number of bucket accesses per insertion of at most a (respectively, d), when given an element per bucket ratio r.

We are now ready to define the *optimal balancing problem*, which is the focus of this paper. Let $\phi : \mathbb{N} \mapsto \mathbb{R}$ be the *cost function* mapping the occupancy of a bucket to its real-valued cost. We assume that ϕ is *non-decreasing* and *convex*. Our goal is to minimize the expected overall cost:

Definition 3: Let O_j be a random variable that counts the number of elements in the j-th bucket. Given γ, a, d and r, the OPTIMAL ACCESS-CONSTRAINED BALANCING PROBLEM consists of finding an $\langle a, d, r \rangle$ balancing scheme that minimizes

$$\phi^{\text{BAL}} = \lim_{m \to \infty} \frac{1}{m} \sum_{j=1}^{m} E\left(\phi\left(O_j\right)\right).$$

Whenever defined, let $\phi_{\text{OPT}}^{\text{BAL}}$ denote its optimal cost.

For example, in the trivial case of the identity cost function $\phi(x) = x$ and no overflow list ($\gamma = 0$), $\phi_{\text{OPT}}^{\text{BAL}}$ corresponds to the average load per bucket, which is exactly r, no matter what insertion algorithm or hash functions are used.

IV. Theoretical Lower Bounds

We next show a lower bound on the achievable value of the optimal cost $\phi_{\text{OPT}}^{\text{BAL}}$, as a function of the number of buckets m, the number of elements n, the average number of bucket accesses a, and the overflow fraction γ.

The lower bound is derived using a modified *offline* setting. In this setting, each bucket access is considered as a distinct element, as if initially $a \cdot n$ distinct elements were hashed to the buckets, using a single hash function each. After storing all elements, we conceptually choose exactly $(a - 1 + \gamma) \cdot n$ of the elements in a way that minimizes the cost function ϕ^{BAL}, resulting in exactly $(1 - \gamma) \cdot n$ elements in the buckets. Since the cost function ϕ is convex, then the marginal cost is the largest in the most occupied buckets. Therefore, a cost-minimizing removal process would remove element by element, picking the next element to remove in the most occupied bucket at each time.

Since we picked these elements in an offline manner, we necessarily perform better than any online setting. Thus, we bound the achievable value of $\phi_{\text{OPT}}^{\text{BAL}}$.

Theorem 1: When all hash functions are uniform, the optimal expected limit balancing cost $\phi_{\text{OPT}}^{\text{BAL}}$ in the OPTIMAL ACCESS-CONSTRAINED BALANCING PROBLEM is lower-bounded by

$$\phi_{\text{LB}}^{\text{BAL}} = \sum_{j=0}^{k_0+1} P_{\text{LB}}\left(i\right) \cdot \phi\left(i\right),$$

where k_0 is the largest integer such that

$$\frac{a \cdot r \cdot \Gamma\left(k_0, a \cdot r\right)}{(k_0 - 1)!} + k_0 \cdot \left(1 - \frac{\Gamma\left(k_0 + 1, a \cdot r\right)}{k_0!}\right) < r\left(1 - \gamma\right),$$

$\Gamma\left(s, x\right) = \int_x^\infty t^{s-1} e^{-t} dt$ is the upper incomplete gamma function, and P_{LB} is a specific distribution that depends only on a, r, and γ, but does not depend on the cost function ϕ.

Specifically, the distribution P_{LB} is defined as follows:

$$P_{\text{LB}}\left(i\right) = \begin{cases} e^{-a \cdot r} \frac{(a \cdot r)^i}{i!} & 0 \leq i < k_0 \\ e^{-a \cdot r} \frac{(a \cdot r)^{k_0}}{k_0!} + e_0 + k_0 + 1 - & \\ \quad k_0 p_0 - p_0 - r \cdot (1 - \gamma) & i = k_0 \\ -e_0 - k_0 + k_0 p_0 + r \cdot (1 - \gamma) & i = k_0 + 1 \\ 0 & \text{otherwise} \end{cases}$$

where $e_0 = \frac{a \cdot r \cdot \Gamma(k_0, a \cdot r)}{(k_0 - 1)!}$, and $p_0 = \frac{\Gamma(k_0 + 1, a \cdot r)}{(k_0)!}$.

Proof Outline: (We remind that full proofs are provided in [9]) As explained, the proof considers an offline setting with $a \cdot n$ elements. We then note that the random variable X that counts the number of elements in a specific bucket after the initial placement of the $a \cdot n$ elements is distributed approximately as Poisson with $\lambda = \frac{a \cdot n}{m}$, whose cumulative distribution function (CDF) is given by $\Pr\{X \leq x\} = \frac{\Gamma(x+1, \lambda)}{x!}$.

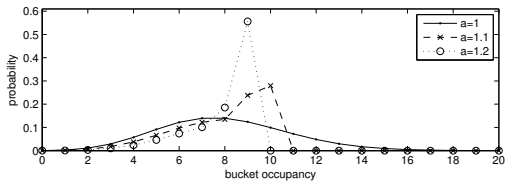


Fig. 2. The probability density function of the distribution P_{LB} over the bucket occupancies. Each distribution P_{LB} induces a lower bound on the balancing cost, given different system constraints. In this figure, we fixed load $r = 8$, overflow fraction $\gamma = 0$, and used different values of average access rate a.

Since elements are then removed one by one from the largest buckets, we intuitively "cut the right side" of the Poisson distribution, up to a precise point k_0 provided by the problem parameters. ■

Interestingly, the element-elimination algorithm does not depend on the precise convex cost function ϕ. This is why the resulting bucket-load distribution P_{LB} is independent of ϕ as well.

In addition, this distribution P_{LB} is defined on a compact space. As a result, it can also be shown that if a sequence of cost functions $\{\phi_k\}$ converges pointwise to some cost function ϕ, then the sequence of lower bounds converges as well to the corresponding lower bound on ϕ. This can then be used to extend the cost functions to the maximum-load metric commonly used in the literature [7], [11], [12].

Fig. 2 shows the lower-bound distribution $P_{\text{LB}}(i)$ for $r = \frac{n}{m} = 8$, $\gamma = 0$ and $a \in \{1, 1.1, 1.2\}$. Note that when $a = 1$, all elements use a single lookup, and therefore there is no element elimination in the offline algorithm. The distribution P_{LB} simply follows a Poisson distribution with parameter $\lambda = r$, as shown using the solid line. Then, for larger values of a, the element elimination algorithm reduces the probability of having a large bin load.

We also consider a setting where $\ell \leq d$ different distributions over the buckets are used by the d hash functions. Denote these distributions by $f^1, \ldots, f^\ell$, and assume that distribution f^i is used by a fraction k_i of the total bucket accesses, with $\sum_{i=1}^{\ell} k_i = 1$. We now show that Theorem 1 holds also in this case when $\sum_{p=1}^{\ell} k_p f_p(i) = \frac{1}{m}$.

Theorem 2: If $\sum_{p=1}^{\ell} k_p f_p(i) = \frac{1}{m}$ then the optimal expected limit balancing cost $\phi_{\text{OPT}}^{\text{BAL}}$ in the OPTIMAL ACCESS-CONSTRAINED BALANCING PROBLEM has the same lower bound as in Theorem 1.

V. SINGLE - A SINGLE-CHOICE BALANCING SCHEME

We have found a lower-bound for the optimal cost. In the sequel, we focus on finding values of a and γ in which we can *match this bound*.

We start by analyzing a simplistic balancing scheme, denoted SINGLE, that is associated with 2 parameters h and p. This scheme only uses a single uniformly-distributed hash function H. Each element is stored in bucket $H(x)$ if it has less than h elements. In case there are exactly h elements, the element is stored in the bucket with probability p and in the overflow list with probability $1 - p$. Otherwise, the element is stored in the overflow list.

In recent years, several balancing schemes have been modeled using a deterministic system of differential equations [10], [13], [14]. We adopt this approach and provide a succinct description of SINGLE in [9]. Then, we solve the system of differential equations yielding the following optimality result.

Theorem 3: Consider the SINGLE balancing scheme with m buckets and n elements, and use the notations of k_0, p_0, e_0 and P from Theorem 1. Then for any value of γ, the SINGLE scheme solves the OPTIMAL ACCESS-CONSTRAINED BALANCING PROBLEM for $a = 1$ whenever it satisfies the two following conditions:
(i) $h = k_0$;
(ii) p is the solution of the following fixed-point equation: $\frac{e^{-p \cdot r}}{(1-p)^h} - \frac{e^{-r}}{(1-p)^h} \sum_{i=0}^{h-1} \frac{(r \cdot (1-p))^i}{i!} = P(k_0)$.

Note that we also double-checked the accuracy of the differential-equation–based model using simulations [9].

VI. SEQUENTIAL - A MULTIPLE-CHOICE BALANCING SCHEME

We now introduce the SEQUENTIAL scheme, which is also associated with two parameters h and p. In the SEQUENTIAL scheme, we use an ordered collection of d hash functions $\mathcal{H} = \{H_1, \ldots, H_d\}$, such that all functions are independent and uniformly distributed. Upon inserting an element x, the scheme successively reads the buckets $H_1(x), H_2(x), \ldots, H_d(x)$, and places x in the first bucket that satisfies one of the following two conditions: *(i)* the bucket stores less than h elements, or, *(ii)* the bucket stores exactly h elements, and x is inserted with probability p. If the insertion algorithm fails to store the element in all the d buckets, x is stored in the overflow list. Last, to keep an average number of bucket accesses per element of at most a, the process stops when a total of $a \cdot n$ bucket accesses has been reached; the remaining elements are placed in the overflow list.

We analyze the SEQUENTIAL scheme by reducing it to the SINGLE scheme: Since both the SINGLE and SEQUENTIAL schemes use the same uniform distribution, a new attempt to insert an element after an unsuccessful previous attempt in the SEQUENTIAL scheme is equivalent to creating a new element in the SINGLE scheme and then trying to insert it. In other words, the number of elements successfully inserted by the SEQUENTIAL scheme after considering n elements and using a total of $a \cdot n$ bucket accesses is the same as the number of elements successfully inserted by the SINGLE scheme after considering $a \cdot n$ elements.

Theorem 4: Consider the SEQUENTIAL balancing scheme with m buckets and n elements, and use the notations of k_0, p_0, e_0 and P from Theorem 1. The SEQUENTIAL scheme solves the OPTIMAL ACCESS-CONSTRAINED BALANCING PROBLEM whenever it satisfies the three following conditions:
(i) $h = k_0$;
(ii) all $a \cdot n$ memory accesses are exhausted before or immediately after trying to insert the n-th element;

(iii) p is the solution of the following fixed-point equation: $\frac{e^{-p\cdot a\cdot r}}{(1-p)^h} - \frac{e^{-a\cdot r}}{(1-p)^h}\sum_{i=0}^{h-1}\frac{(a\cdot r\cdot(1-p))^i}{i!} = P(k_0)$.
Moreover, the optimality region is given by the overflow list of size $\gamma_0 \cdot n$ that results in exhausting all $a \cdot n$ memory immediately after trying to insert the n-th element.

VII. The Multi-Level Hash Table (MHT) Balancing Scheme

The *multi-level hash table* (MHT) balancing scheme conceptually consists of d separate subtables $T_1, \ldots, T_d$, where T_i has $\alpha_i \cdot n$ buckets, and d associated hash functions $H_1, \ldots, H_d$, defined such that H_i never returns values of bucket indices outside T_i.

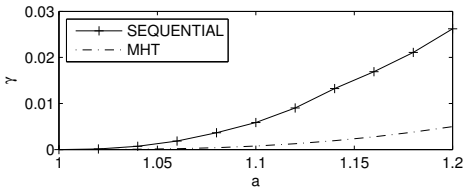
Using the MHT scheme, element x is placed in the smallest i that satisfies one of the following two conditions: *(i)* the bucket $H_i(x)$ stores less than h elements, or, *(ii)* the bucket $H_i(x)$ stores exactly h elements, and element x is then inserted with probability p. If the insertion algorithm fails to store the element in all the d tables, x is placed in the overflow list. Since that smallest i with available space is used, the bucket accesses for each element x are sequential, starting from $H_1(x)$ until a place is found or all d hash functions are used (and the element is stored in the overflow list).

We skip to the optimality theorem of the MHT scheme.

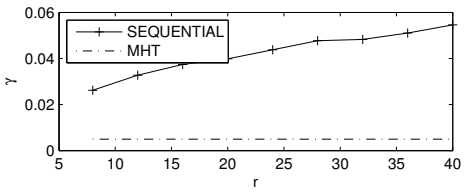
Theorem 5: Consider an $\langle a, d, r\rangle$ MHT balancing scheme in which each subtable T_j has $\alpha_j \cdot m$ buckets, with $\sum \alpha_j = 1$, and use the notations of k_0, p_0, e_0 and P from Theorem 1. Further, let $p(a)$ denote the overflow fraction of the SINGLE scheme with $a \cdot n$ elements (that is, $p(a) = \gamma^t_{\text{SINGLE}}(1, m, a\cdot n, h, p)$, see [9]). Then, the $\langle a, d, r\rangle$ MHT scheme solves the OPTIMAL ACCESS-CONSTRAINED BALANCING PROBLEM whenever it satisfies the following four conditions:
(i) $h = k_0$;
(ii) all $a \cdot n$ memory accesses are exhausted before or immediately after trying to insert the n-th element;
(iii) p is the solution of the following fixed-point equation: $\frac{e^{-p\cdot a\cdot r}}{(1-p)^h} - \frac{e^{-a\cdot r}}{(1-p)^h}\sum_{i=0}^{h-1}\frac{(a\cdot r\cdot(1-p))^i}{i!} = P(k_0)$;
(iv) the subtable sizes $\alpha_j \cdot m$ follow a geometric decrease of factor $p(a)$: $\alpha_j = \left(\frac{1-p(a)}{1-p(a)^d}\right)p(a)^{j-1}$.
Moreover, the optimality region is given by the overflow list of size $\gamma\cdot n$ that results in exhausting all $a\cdot n$ memory immediately after trying to insert the n-th element. Furthermore, if all four conditions are met then all buckets have an identical occupancy distribution.

VIII. Comparative Evaluation and Analysis

Fig. 3(a) shows the optimality region of SEQUENTIAL and MHT with element-per-bucket ratio $r = 8$, and $d = 3$ hash functions. For each value of the average number of bucket accesses a, it shows the minimum value of the overflow fraction γ that suffices to solve the OPTIMAL ACCESS-CONSTRAINED BALANCING PROBLEM. For instance, we can see that for $a \approx 1.1$, SEQUENTIAL achieves optimality for an overflow fraction equal to or larger than approximately 1%. We will show in Section II that the optimal solution for such parameters dramatically reduces the balancing cost compared



(a) Fixed load $r = 8$, worst-case number of memory accesses $d = 3$; variable average memory accesses rate a



(b) Fixed average and worst-case memory accesses rate $a = 1.2, d = 3$; variable load r

Fig. 3. The overflow fraction γ induced by applying SEQUENTIAL and MHT schemes for different system parameters. As expected, MHT requires smaller overflow fraction than SEQUENTIAL.

to $a = 1$ with no overflow list, and therefore improves the performance of the resulting Bloom Filter. In addition, Fig. 3(b) shows the optimality region of SEQUENTIAL and MHT with $a = 1.2$ and $d = 3$ for different values of r. We can see that MHT scales better to higher loads.

IX. Analysis of the Balanced Bloom Filter

Blocked Bloom Filters [4], [5] were the first try to design an access-efficient Bloom Filters. They constrain the k hashed bits to be located in the same memory block, thus causing a single memory access. The blocked Bloom Filter mechanism can be modeled using the SINGLE scheme with no overflow list, that is, $\gamma = 0$, and a cost function ϕ, where $\phi(i)$ expresses the false positive rate incurred to an element in a given memory block given that i elements are hashed to this memory block. More precisely, $\phi(i) = \left(1 - \left(1 - \frac{1}{B}\right)^{ki}\right)^B \approx \left(1 - e^{-\frac{ki}{B}}\right)^k$, where B is the size in bits of the memory block and k is the number of hash functions used. However, although the SINGLE scheme is optimal, its average number of memory accesses is $a = 1$, thus it achieves poor balancing of the elements resulting in a high false positive rate. In this section we suggest to *use our optimal online schemes to achieve a better balancing between the memory blocks, and therefore, a better false positive rate.*

Assuming memory blocks of size B bits and bits-per-element ratio β, the number of elements per bucket r is B/β. Using the optimal online balancing schemes to implement a Bloom Filter requires saving some $b = \lceil \log_2(k_0 + 2)\rceil$ bits in every memory block to count the elements hashed into each one. Thus, $\phi(i) = \left(1 - e^{-\frac{ki}{B-b}}\right)^k$.

In the standard Bloom Filter, the optimal false positive rate is achieved when using $k = r \cdot \ln 2$ hash functions [2].

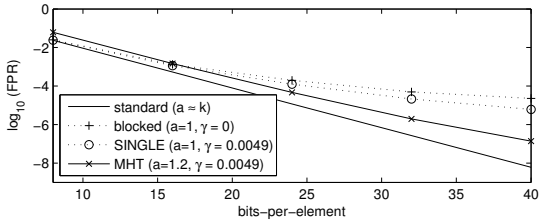


Fig. 4. False positive rate of different Bloom Filter schemes with memory block size of 256 bits and different values of r. SINGLE and MHT use overflow fraction $\gamma = 0.49\%$. In larger value of r, MHT achieves better FPR than SINGLE by two order of magnitude better, albeit with 20% more memory accesses on average.

Although this may not be the best choice in our settings, we will use the same k for simplicity.

Balancing schemes that read more than one memory block on a query operation increase the false positive rate, because a false positive may result from a query operation on each one of the buckets. Thus, since the probability of false positive in every memory block is relatively small, then the overall false positive probability, i.e. the false positive rate, is modeled by $\sum_{j=1}^{d} P_j \cdot \text{FPR}_j$, where P_j is the probability that a query operation results in at least j memory blocks, and FPR_j is the false positive probability of the j-th memory block read.

Fig. 4 compares the false positive rate for different values of bits-per-element ratios with memory block size $B = 256$. The SINGLE balancing scheme performs slightly better than the blocked Bloom Filter scheme. For low values of bits-per-element ratio, the MHT scheme performs worse. This is due to the need to check multiple memory blocks (up to d) on a query operation. However, for larger values of bits-per-element ratios, the MHT performs better by two orders of magnitude, and only one order of magnitude worse than the standard Bloom Filter, which uses an access-inefficient scheme with $a \approx k$ accesses. For example, for a bit-per-element ratio of 24, $k = 17$ hash functions are used, introducing up to 17 memory-read operations in the standard Bloom Filter, which can clearly present memory-throughput and power-consumption issues.

X. Related Work

In Section I, we surveyed the relevant work related to Bloom Filters, we now survey the relevant work related to the classic balancing problems, which were extensively investigated in the last decades for various applications involving allocations of resources [15]. Prime examples are task balancing between many machines [16], [17], item distribution over several locations [18], bandwidth allocation in communication channels [15] or within switches and routers [19], and hash-based data structures [20].

Our paper is most related to the *sequential static multiple-choice balls-and-bins problem* described above. This model was first considered in the seminal work of Azar, Broder, Karlin, and Upfal [7], and had a large impact on modern algorithms and data structures (see surveys in [14], [21]). Note that most papers considered the *maximum load* of the system, while our paper considers the entire load distribution (including the maximum load).

Access-constrained hash-schemes were also considered in [10], which, however, did not consider a cost minimization problem and cannot deal with infinite-size bins.

XI. Conclusion

In this paper we presented an access-efficient variant of Bloom Filters. In this variant, each element is first assigned to a memory block, where a local Bloom Filter is maintained. Our basic approach was two-folded. First, we proposed to maintain load-balancing schemes: a simple approach (using only a single hash function), a greedy approach, and the *mutli-level hash table*. And second, we proposed the usage of an overflow-list that can store elements that are hashed to overloaded buckets. To study this problem, we presented an access-constrained balancing problem and show that, depending on the specific given parameters, our proposed load-balancing schemes are optimal.

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, pp. 485–509, 2004.
[3] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "Beyond Bloom filters: from approximate membership checks to approximate state machines," in *SIGCOMM*, 2006, pp. 315–326.
[4] F. Putze, P. Sanders, and J. Singler, "Cache-,hash- and space-efficient Bloom filters," in *Workshop on Exp. Algorithms*, 2007, pp. 108–121.
[5] Y. Qiao, T. Li, and S. Chen, "One memory access bloom filters and their generalization." in *INFOCOM*, 2011, pp. 1745–1753.
[6] N. L. Johnson and S. Kotz, *Urn models and their application: an approach to modern discrete probability theory*. Wiley NY, 1977.
[7] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal, "Balanced allocations," in *ACM STOC*, 1994, pp. 593–602.
[8] A. Kirsch, M. Mitzenmacher, and U. Wieder, "More robust hashing: Cuckoo hashing with a stash," in *ESA*, 2008, pp. 611–622.
[9] Y. Kanizo, D. Hay, and I. Keslassy, "Access-efficient balanced bloom filters," Comnet, Technion, Israel, Technical Report TR11-07, 2011. [Online]. Available: http://comnet.technion.ac.il/ isaac/papers.html
[10] ——, "Optimal fast hashing," in *INFOCOM*, 2009, pp. 2500–2508.
[11] B. Vöcking and M. Mitzenmacher, "The asymptotics of selecting the shortest of two, improved," in *Analytic Methods in Applied Probability*, 2002, pp. 165–176.
[12] B. Vöcking, "How asymmetry helps load balancing," in *IEEE FOCS*, 1999, pp. 131–141.
[13] A. Kirsch and M. Mitzenmacher, "The power of one move: Hashing schemes for hardware," in *INFOCOM*, 2008, pp. 565–573.
[14] M. Mitzenmacher, A. Richa, and R. Sitaraman, "The power of two random choices: A survey of techniques and results," in *Handbook of Randomized Computing*, 2000, pp. 255–312.
[15] Y. Azar, "On-line load balancing," *Theoretical Computer Science*, pp. 218–225, 1992.
[16] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts, "On-line load balancing with applications to machine scheduling and virtual circuit routing," in *ACM STOC*, 1993, pp. 623–631.
[17] R. Graham, "Bounds for certain multiprocessing anomalies," *Bell System Technical Journal*, vol. 45, pp. 1563–1581, 1966.
[18] B. Godfrey, K. Lakshminarayanan, S. Surana, R. M. Karp, and I. Stoica, "Load balancing in dynamic structured P2P systems," in *INFOCOM*, 2004.
[19] I. Keslassy, "The load-balanced router," Ph.D. dissertation, Stanford Univ., 2004.
[20] G. H. Gonnet, "Expected length of the longest probe sequence in hash code searching," *J. ACM*, vol. 28, no. 2, pp. 289–304, 1981.
[21] A. Kirsch, M. Mitzenmacher, and G. Varghese., *Hash-Based Techniques for High-Speed Packet Processing*. DIMACS, 2010, ch. 9, pp. 181–218.