The power of 1+α for Memory Efficient Bloom Filters

Evgeni Krimer and Mattan Erez

Electrical and Computer Engineering Department
University of Texas at Austin
{krimer,mattan.erez}@mail.utexas.edu

Abstract. This paper presents a cache-aware Bloom Filter algorithm with improved cache behavior and a lower false positive rates compared to prior work. The algorithm relies on the power-of-two choice principle to provide a better distribution of set elements in a Blocked Bloom Filter. Instead of choosing a single block, we insert new elements into the least-loaded of two blocks to achieve a low false-positive rate while performing only two memory accesses on each insert or query operation. The paper also discusses an optimization technique to trade off cache effectiveness with the false-positive rate to fine-tune the Bloom Filter properties.

1 Introduction

The *Bloom Filter* (BF) is a space-efficient data structure representing a set, which provides *add* and *probabilistic membership query* operations with a certain rate of false positives, the *False Positive Rate*, and no false negatives [1]. *Bloom Filters* are implemented in either hardware or software and are commonly used in various fields including *networks* [2] and web services [3]. The two main metrics of a *Bloom Filter* are its performance (throughput and latency) and its *False Positive Rate* (FPR). Our work addresses both at the same time by offering the *Power-of-*$1+\alpha$ *Blocked Bloom Filter* scheme, which introduces a parametrizable tradeoff between performance and the FPR.

The performance of the BF often has a large impact on overall system performance. For example, BF query latency and bandwidth dictate the performance of WebCache servers, which implement a software BF [3], and network routers, which use a hardware BF [4]. As shown in Section 2.1, BF query operations require k memory accesses to random locations, which stress the cache in software implementations and add cost and time to hardware BFs. By changing the algorithm to exploit *locality* (the likelihood of the system to reuse recently used data [5]), the performance of BFs can be improved. For software implementations, increasing *locality* will decrease the number of external memory accesses due to a higher number of *cache hits*. For hardware BFs, locality enables increased lookup capacity by utilizing smaller independent blocks to construct the memory array [4]. Due to this duality of software and hardware implementations, throughout this paper, we use the term *block* to represent a cache line, memory page, or memory bank, depending on the specific implementation.

The FPR of the BF also directly impacts overall system performance and behavior, because queries that are falsely answered as belonging to the set typically result in unnecessary costly work

[1] This is a preprint of an article whose final and definitive form has been published in the Internet Mathematics

being performed. Thus, it is not enough to improve locality and performance, but the FPR must be controlled so that it remains within a reasonable bound of the classic BF algorithm.

This work explores a smooth tradeoff space between the performance and the FPR. Previous attempts of improving locality have so far been either too aggressive or too conservative and have not reached an overall optimal point of balancing performance and the FPR. In work motivated by a hardware implementation of a BF for a network router [6], the authors suggest to pair hash functions such that even hash functions can choose any bit in the array but odd hash functions are restricted to choosing a bit in the same block as their corresponding even pair. Thus only $\frac{k}{2}$ blocks are accessed for each BF operation. Their simulation results show a minor effect on the FPR. On the other hand, the algorithm they present is still very conservative with respect to locality and still requires $\mathcal{O}(k)$ ($\frac{k}{2}$ to be exact) memory block accesses.

A different publication, from the perspective of a software BF implementation, presents the *Blocked Bloom Filter* [7]. This extremely aggressive approach to exploit locality requires only a single block access for every query operation. However, it significantly increases the FPR for some BF configurations, as we discuss in this paper.

Our approach offers a flexible and parametrizable BF algorithm for exploiting locality and we summarize our main contributions below:

- We analyze the FPR of the *Blocked Bloom Filter* in detail and show how its worse FPR, relative to the classic BF algorithm, is a result of a poor distribution of elements between memory blocks.
- We present the *Power-of-Two Blocked Bloom Filter*, which improves the load balance of elements across memory blocks, thus minimizing the FPR increase relative to the classic BF algorithm for some configurations. We then derive and validate an analytical model for the FPR of the *Power-of-Two Blocked Bloom Filter*.
- We develop the *Power-of-1+α Blocked Bloom Filter*, which enables a smooth tradeoff curve between the locality and the FPR by combining the *Blocked Bloom Filter* and the *Power-of-Two Blocked Bloom Filter* approaches. We also derive and validate an analytical model for the FPR of the *Power-of-1+α Blocked Bloom Filter*.
- We show how to fine-tune the tradeoff options in general, and how to find the optimal mix of Power-of-Two and Blocked Bloom Filters for any given BF configuration.

The rest of the paper is organized as follows. In Section 2, we describe previously-published algorithms of the *(Classic) Bloom Filter* and the *Blocked Bloom Filter* discussing their performance and FPR metrics. In Section 3.1, we propose the *Power-of-Two Blocked Bloom Filter* along with its analytical model as an alternative to the *Blocked Bloom Filter*. Then, in Section 3.2, we present the *Power-of-1+α Blocked Bloom Filter* that allows a fine-grained tuning to optimize the performance – FPR tradeoff; we provide an analytical model for the *Power-of-1+α Blocked Bloom Filter* algorithm. In Section 4, we compare the simulation results of all mentioned approaches in terms of the FPR and distribution load balance. We discuss the results and provide experimental evaluations of the optimal α parameter for different configurations of the BF. Finally, we summarize our findings and present plans for future work in Section 5.

2 Background & Related Work

The *Bloom Filter* is a relatively simple, yet efficient, probabilistic data structure that represents a set. In this section we begin with a brief overview of the algorithm and the FPR properties of

the *(Classic) Bloom Filter*. Many published modifications to this classic BF exist in the literature, including one that uses the same *power-of-two* principle that is the basis of this paper [8]. Most of this prior work, however, including [8], does not address the performance (throughput/latency) of the BF. We limit the discussion in this paper to the aggressive Blocked BF [7] (Section 2.2), which significantly improves performance at the cost of a worse FPR. We omit the details of the very conservative approach to improve performance presented in [6] as it only reduces the number of accesses by a factor of two, still requiring $\mathcal{O}(k)$ accesses.

2.1 The (Classic) Bloom Filter

This subsection provides a brief overview of the *Bloom Filter* data structure, algorithm, and FPR derivation. The BF uses a vector of m bits, which represents the set of elements S (where $\|S\| = n$) and allows membership queries. The BF utilizes k hash functions ($h_1..h_k$), with each function mapping a potential set element to a single bit location in the range of $[0, m)$.

Algorithm 1 Classic Bloom Filter — *k block accesses on average*

```
function Add(x)
  for i = 0 to k do
    mem(h_i(x)) ⇐ 1
  end for
end
```

```
function Query(x)
  for i = 0 to k do
    if mem(h_i(x)) == 0 then
      return NOT FOUND
    end if
  end for
  return FOUND
end
```

Table 1: Definition of symbols used.

n	number of elements
c	bits per element
m	total memory size ($\equiv c \cdot n$) [bits]
B	block size [bits]
b	number of blocks ($\equiv m/B$)
k	number of hash functions
h_i	hash function selecting a bit out of the bit array : $elem \Rightarrow [0..m)$
hb_i	hash function selecting a bit out of a block : $elem \Rightarrow [0..B)$
g_i/g	hash function selecting a block : $elem \Rightarrow [0..b)$
$mem(x)$	bit x of the bit array
$mem[y](x)$	bit x of the block y

As shown in Algorithm 1 (and using the symbols summarized in Table 1), adding an element is done by setting all the bits pointed to by the k hash functions for this element to 1. Note that some of these bits might already be set, because they were touched by prior add operations of other elements. To query a membership of an element, all the bits pointed by the results of the k hash functions are tested. Only if all of them are set, the query result is positive. However, a false positive response is possible because all the relevant bits might be have been set while adding unrelated elements. Thus a query result may be positive even when the element queried was never

added to the BF. Using basic probability properties and algebra (refer to [9] for more detail), the FPR of the *(Classic) Bloom Filter* can be written as:

$$FPR_{Basic}(m,n,k) = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-\frac{kn}{m}}\right)^k \quad (1)$$

Performance-wise, the worst-behaving query operation would require accessing k different blocks, assuming there are at least k blocks in the BF bit array. In practice, Lemma 1 proves that an average query would require accessing $k - \mathcal{O}(\frac{1}{b})$ different blocks, and thus the average case quickly approaches the worst case performance.

2.2 The Blocked Bloom Filter

The *Blocked Bloom Filter* attempts to improve query performance by exploiting *locality* [7] . As described below, Algorithm 2 associates a single block of B bits with each element. In this way, each element is confined to a single block with different elements associated with different blocks, and potentially multiple elements associated with each block. As a result, it allows us to perform add/query operations with a single block access.

Algorithm 2 Blocked Bloom Filter — *1 block access*

```
function Add(x)
  block ⇐ g(x)
  for i = 0 to k do
    mem[block](hb_i(x)) ⇐ 1
  end for
end
```

```
function Query(x)
  block ⇐ g(x)
  for i = 0 to k do
    if mem[block](hb_i(x)) == 0 then
      return NOT FOUND
    end if
  end for
  return FOUND
end
```

The FPR of the *Blocked Bloom Filter* is strongly dependent on load balancing among the blocks. Let $D_{Blocked}(j)$ be the probability of having exactly j elements in a block; then, the FPR can be represented as [7]:

$$FPR_{Blocked}(B,k) = \sum_{j=0}^{n} D_{Blocked}(j) \cdot FPR_{Basic}(B,j,k) = \sum_{j=0}^{\infty} D_{Blocked}(j) \cdot FPR_{Basic}(B,j,k) \quad (2)$$

An element has an equal probability of $\frac{1}{b}$ to be associated with a block. Therefore, the distribution of elements among the blocks can be modeled using a *Binomial* distribution with parameter $p = \frac{1}{b}$ for n elements : $D_{Blocked}() = Binomial(n, \frac{1}{b})$. Since $\frac{n}{b}$ is a small constant, and $\frac{n}{b} \equiv \frac{B}{c}$, a *Poisson* approximation can be used [7]:

$$D_{Blocked}() = Binomial(n, \frac{1}{b}) \approx Poisson(\frac{n}{b}) \equiv Poisson(\frac{B}{c}) \quad (3)$$

As proved by Lemma 2, however, in the special ideal case of a fully-balanced distribution among the blocks, $FPR_{Blocked\_Balanced} = FPR_{Basic}$. This observation, along with Equation 2, leads us to

conclude that improving the balance among the blocks will improve the FPR whose lower bound will be the FPR_{Basic} in case of a fully-balanced distribution.

3 Proposed Algorithms

In this section, we present two novel BF modifications. Using the *power-of-two* principle, we improve the load balance among the blocks in order to reduce the FPR per our conclusion from Section 2.2 and introduce the *Power-of-Two Blocked Bloom Filter*. Then, we present the *Power-of-1+α Blocked Bloom Filter*, which is a combination of the *Power-of-Two Blocked Bloom Filter* with the existing Blocked BF that provides a smooth tradeoff between the FPR and performance. For both algorithms, we develop an analytical model of the FPR and validate them later in Section 4.

3.1 The Power-of-Two Blocked Bloom Filter

The distribution of elements among the blocks is similar to a problem of random distribution of n balls into b bins. The *power-of-two* idea described in literature [9], suggests that instead of placing the ball into a randomly selected bin, we should select $d \geq 2$ bins and choose the least-loaded among them. This approach would decrease the number of the balls in the bin that contains the most from $\Theta\left(\frac{ln\ n}{ln(ln\ n)}\right)$ to $\Theta\left(ln\left(ln\ n\right)\right)$.

We propose a modification to the Blocked BF that uses two hash functions, g_1 and g_2, to select two blocks for each element. For the *add* operation, we choose the least loaded block out of the two, and the element is added to the chosen block in the same way as in the Blocked BF. To perform a membership query, we have to check both blocks that are associated with the element, because we do not know which of the two blocks the element was added to. Thus, the result will be a positive if any of the two blocks returns a positive. In other words, only if the query result is negative for both blocks, the outcome will be negative. Algorithm 3 presents the algorithm.

Algorithm 3 Power-of-Two Blocked Bloom Filter — *2 block accesses*

```
function Add(x)
    block ⇐ choose_less_loaded_block(g1(x), g2(x))
    for i = 0 to k do
        mem[block](hbi(x)) ⇐ 1
    end for
end
```

```
function Query(x)
    block1 ⇐ g1(x)
    block2 ⇐ g2(x)
    for i = 0 to k do
        if mem[block1](hbi(x)) == 0 then
            for j = 0 to k do
                if mem[block2](hbj(x)) == 0 then
                    return NOT FOUND
                end if
            end for
            return FOUND
        end if
    end for
    return FOUND
end
```

Making the algorithm access two blocks and selecting the least-loaded block, improves the load balance among the blocks (see Figure 1). Equation 4 shows the FPR of the Power-of-Two Blocked

BF, which can be derived in a similar fashion to the FPR of the Blocked BF (Equation 2), with two important changes. First, the distribution $D_{PTwo}(x)$ is different and more balanced than the distribution of the Blocked BF - $D_{Blocked}(x)$. Second, unlike the Blocked BF, querying the Power-of-Two Blocked BF selects two blocks and queries each of them independently, and hence the FPR requires a coefficient of 2.

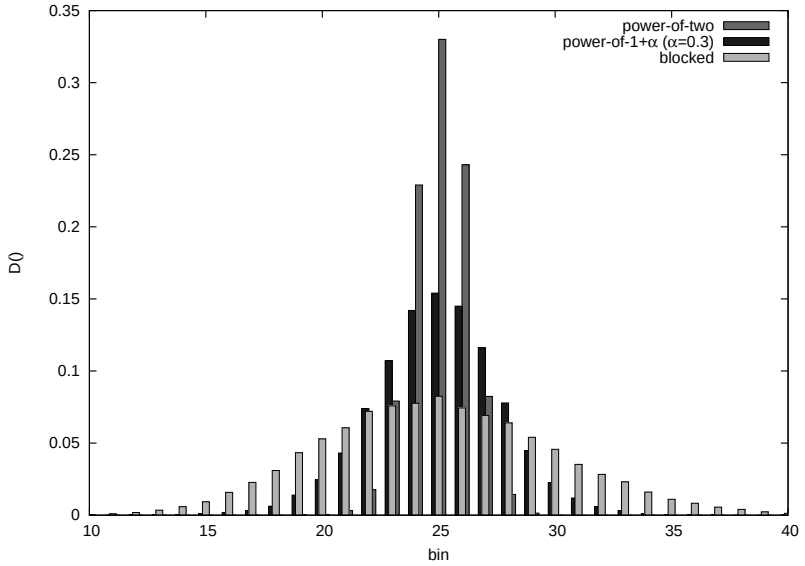

Fig. 1: $D()$ distribution of various approaches after inserting n=10^6 elements with c=20bits per element.

$$FPR_{PTwo}(B,k) = 2 \cdot \sum_{j=0}^{\infty} D_{PTwo}(j) \cdot FPR_{Basic}(B,j,k) \tag{4}$$

The distribution of elements among the blocks using the Power-of-Two Blocked BF, $D_{PTwo}(x)$, differs from the one generated by the Blocked BF, $D_{Blocked}(x)$ as depicted in Figure 1. Therefore, $FPR_{PTwo}(B,k) \neq 2FPR_{Blocked}(B,k)$. Moreover, as it will be discussed and shown in Section 4, FPR_{PTwo} is actually lower than $FPR_{Blocked}$ for a range of configurations.

The distribution $D_{PTwo}(x)$ is derived iteratively by analyzing the process of adding elements to the Power-of-Two Blocked BF structure. After adding an element, there are three possible outcomes: the number of blocks having x elements can increase by 1 if the element was added to a block with $x-1$ elements (so that it becomes a new block with x elements); it can decrease by 1 if an element was added to a block with x elements (that becomes a block with $x+1$ elements), or finally, it can remain unchanged.

The algorithm selects two blocks and chooses the least-loaded one. In order to choose the block with exactly x elements, either both selected blocks, should contain x elements or one block contains x elements and the second block any greater number. The order of selecting the blocks is unimportant. Let $P_{choose}(x)$ be the probability of choosing to add the new element to a block with exactly x elements. Then, we can derive $P_{inc}(x)$ as the probability to increase the number

of blocks with exactly x elements and $P_{dec}(x)$ as the probability to decrease the number of blocks with exactly x:

$$P_{choose}(x) = (D_{PTwo}(x))^2 + 2 \cdot D_{PTwo}(x) \cdot \sum_{j=x+1}^{\infty} D_{PTwo}(j) \tag{5}$$

$$P_{inc}(x) = P_{choose}(x-1)$$

$$P_{dec}(x) = P_{choose}(x)$$

Let $N_n(x)$ be the number of blocks with x elements after the n^{th} element insertion, we can derive $D_{PTwo}(x)$:

$$D_{PTwo}(x) = \frac{N(x)}{b}$$

$$N_{n+1}(x) = N_n(x) + 1 \cdot P_{inc}(x) - 1 \cdot P_{dec}(x)$$

$$\frac{\partial N(x)}{\partial n} = P_{inc}(x) - P_{dec}(x)$$

$D_{PTwo}(x)$ is derived in Equation 6. We solve it numerically to obtain $FPR_{PTwo}(B,k)$ using Equation 4. These results are validated and compared with other algorithms later in Section 4.

$$\frac{\partial D_{PTwo}(x)}{\partial n} = \frac{1}{b} \cdot \frac{\partial N(x)}{\partial n} = \frac{1}{b} \cdot (P_{choose}(x-1) - P_{choose}(x)) \tag{6}$$

3.2 The Power-of-1 + α Blocked Bloom Filter

In this subsection, we introduce the *Power-of-*$1+\alpha$ *Blocked Bloom Filter*. This approach allows fine-grained control over the distribution balance, the FPR, and the number of blocks to be accessed by combining the Blocked BF with the Power-of-Two Blocked BF.

The tradeoff is controlled using the $\Phi(x)$ function, which uses $0 \leq \alpha \leq 1$ defined as follows:

$$\Phi(x) = \begin{cases} 0 & \textit{with probability } 1-\alpha \\ 1 & \textit{with probability } \alpha \end{cases}$$

Note that although $\Phi(x)$ is a probabilistic function, it is deterministic, i.e. a result for a given element never changes. Algorithm 4 shows the usage of $\Phi(x)$ in the *add* and *query* operations.

Deriving $FPR_{\alpha}(B,k)$ is similar to the derivation of $FPR_{PTwo}(B,k)$ (Equation 4), although the coefficient will be $1+\alpha$ rather than 2 and the distribution is different as well, as shown in Equation 7. The probability of using the Power-of-Two Blocked BF algorithm with a coefficient of two is α. On the other hand, the probability of using the Blocked BF algorithm with a coefficient of one is $1-\alpha$. One can observe that for $\alpha = 0$, this scheme reduces to the Blocked BF, whereas for $\alpha = 1$, it turns into the Power-of-Two Blocked BF.

Algorithm 4 Power-of-1 + α Bloom Filter — *1+α block accesses on average*

function Add(x)	**function** Query(x)
if $\Phi(x)$ **then**	**if** $\Phi(x)$ **then**
$Add_{Power-of-Two}(x)$	**return** $Query_{Power-of-Two}(x)$
else	**else**
$Add_{Blocked}(x)$	**return** $Query_{Blocked}(x)$
end if	**end if**
end	**end**

$$\begin{aligned} FPR_\alpha(B,k) &= \alpha \cdot \left[2 \cdot \sum_{j=0}^{\infty} D_\alpha(j) \cdot FPR_{Basic}(B,j,k) \right] + (1-\alpha) \cdot \left[\sum_{j=0}^{\infty} D_\alpha(j) \cdot FPR_{Basic}(B,j,k) \right] \\ &= (1+\alpha) \cdot \sum_{j=0}^{\infty} D_\alpha(j) \cdot FPR_{Basic}(B,j,k) \end{aligned} \tag{7}$$

$D_\alpha(x)$ can be calculated iteratively. Adding an element to the data structure, with probability of α will follow the Power-of-Two Blocked BF algorithm and update the distribution in a way similar to Equation 6. As before, with probability of $1-\alpha$, the add operation will follow the Blocked BF algorithm and update the distribution according to Equation 10. Lemma 3 proves that the distribution defined by Equation 3 is the solution of the iterative Equation 10. To simplify the equation, we define $P_{\alpha\_choose}(x)$ similarly to $P_{choose}(x)$ in Equation 5 and use it in Equation 8. We solve Equation 8 numerically to obtain $FPR_\alpha(B,k)$ using Equation 7. These results are validated and compared with other algorithms later in Section 4.

$$P_{\alpha\_choose}(x) = (D_\alpha(x))^2 + 2 \cdot D_\alpha(x) \cdot \sum_{j=x+1}^{\infty} D_\alpha(j)$$

$$\frac{\partial D_\alpha}{\partial n} = \alpha \cdot \left[\frac{1}{b} \cdot (P_{\alpha\_choose}(x-1) - P_{\alpha\_choose}(x)) \right] + (1-\alpha) \cdot \left[\frac{1}{b} \left(D_\alpha(x-1) - D_\alpha(x) \right) \right] \tag{8}$$

4 Results and Discussion

This section compares the FPR and distribution metrics of the schemes described in our work. Results are generated using a simulator written in C++ and run on an Intel® Core™2 CPU. We use the linear congruential algorithm with 48-bit integer arithmetic (drand48) to generate pseudo-random numbers to be used as the hash functions for adding elements. Each simulation is repeated 100 times, and the FPR is derived from the number of set bits after 10^6 random elements are added. Table 2 summarizes the parameters used in the simulations.

In Section 2.2, we mentioned that improving balance among blocks improves the FPR. The Power-of-Two Blocked BF uses the *power-of-two principle* to achieve this. In the Power-of-1+α Blocked BF approach we introduce α to control the balance. Figure 1 shows experimental results for the distributions $D(x)$ of all three approaches. The benefit of the *power-of-two principle* is

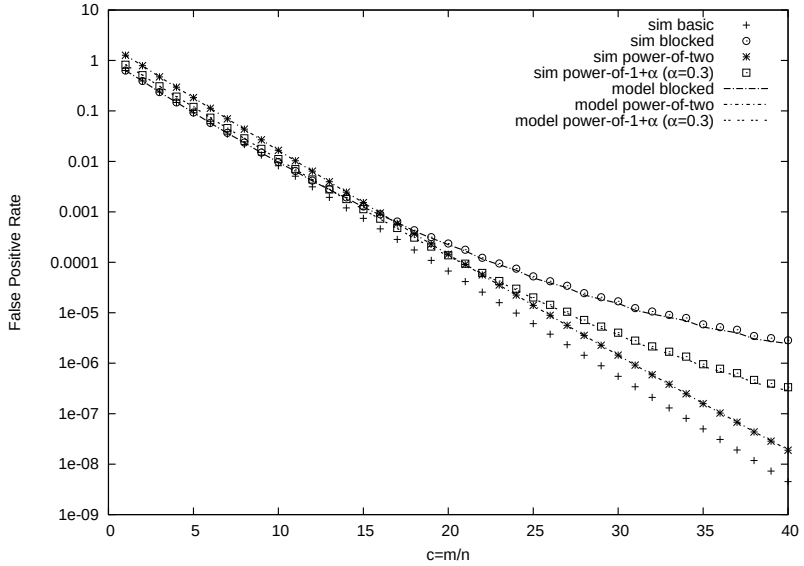

Fig. 2: False Positive Rates of different Bloom Filter approaches for varying values of c (bits per element).

Table 2: Parameters used for simulation.

n	number of elements	10^6
c	bits per element	(1..40)
m	total memory size	$\equiv c \cdot n$ [bits]
B	block size	500 [bits]
b	number of blocks	$\equiv m/B$
k	number of hash functions	$\lvert ln2 \cdot c \rvert_{round}$

clearly seen where the $1 + \alpha$ approach (with $\alpha = 0.3$), as expected, is in between the the Blocked and Power-of-Two Blocked BFs.

The FPRs of the different approaches are depicted in Figure 2. It clearly shows the disadvantage of the Blocked BF against the Power-of-Two Blocked BF for configurations with $c \geq 17$. The figure also shows how the Power-of-1+α Blocked BF ($\alpha = 0.3$) enables greater flexibility and yields the best overall FPR for $13 \leq c \leq 20$. Finally, Figure 2 validates the analytical models and shows a good match with the simulation results for all three schemes.

The parameter α in the Power-of-1+α Blocked BF controls the algorithm. As mentioned before, both the Blocked BF and the Power-of-Two Blocked BF are special cases of the Power-of-1+α Blocked BF for $\alpha = 0$ and $\alpha = 1$ respectively. Therefore, to choose the best scheme for each BF parameter configuration, an optimal α value should be selected.

Figure 3 shows the ratio of the Power-of-1+α Blocked BF FPR versus the Blocked BF FPR as a function of α for different configurations of c. This function has a convex form where an optimal α exists. As an example, Figure 3 shows the optimal points for c=16, 18, 20, which are respectively α=0.3, 0.4, 0.5.

Figure 4 plots the optimal α values across a range of BF configurations (varying c). The figure clearly shows that for $c \leq 10$, the best FPR is achieved with $\alpha = 0$, i.e. the Blocked BF; the Power-

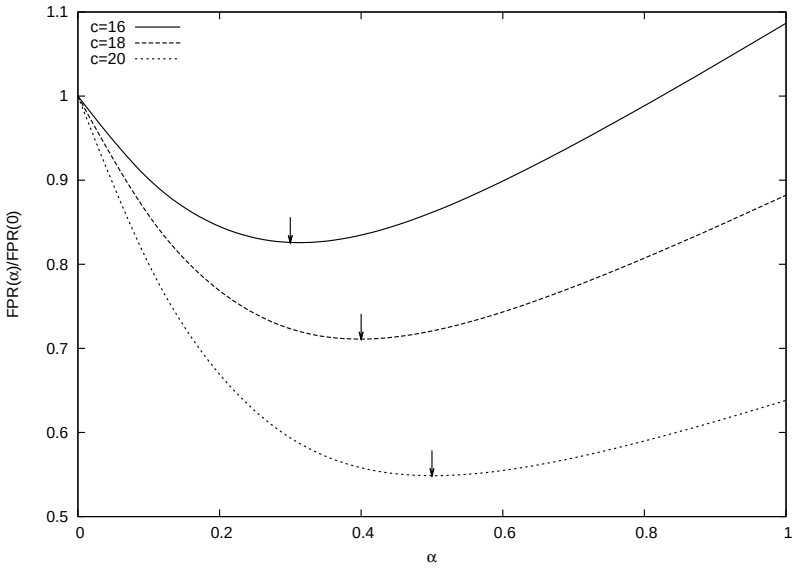

Fig. 3: $\frac{FPR(\alpha)}{FPR(0)}$ for different values of c (16,18,20).

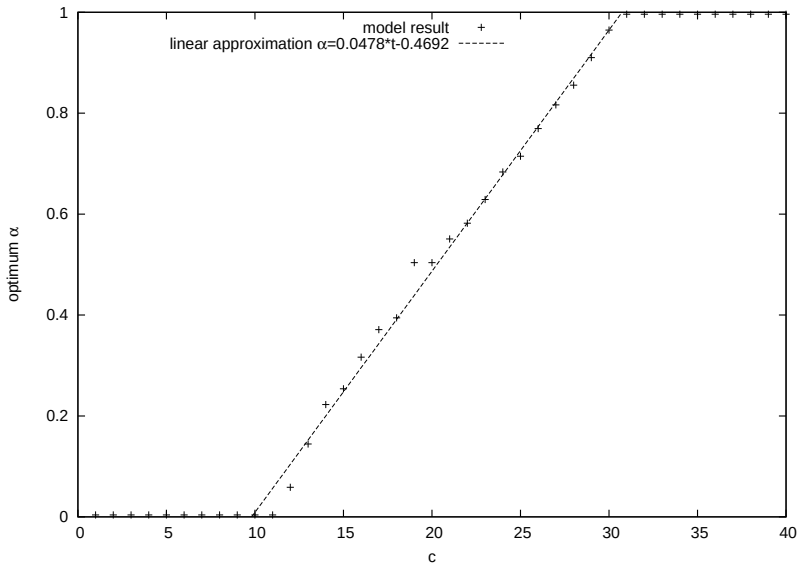

Fig. 4: Optimum α for different values of c can be linearly approximated.

of-Two Blocked BF is optimal for $c \geq 31$; and the optimal α in between can be approximated linearly.

5 Summary and Future Work

The *Bloom Filter* is an efficient and widely used probabilistic data structure implemented both in software and hardware. In Section 2.1, we summarized the (Classic) BF algorithm, derived its FPR, and showed how its performance disadvantaged due to accesses to multiple (k) memory blocks. We then described the Blocked BF [7], which is a high-performance alternative, which only requires a single memory block access. The Blocked BF, however, introduces a higher *False Positive Rate* (FPR). We discussed the reason for the FPR increase and showed that by balancing the load of elements among the blocks, the FPR could be improved.

We followed this conclusion and developed the Power-of-Two Blocked BF, which improves the balance of element distribution using the *power-of-two choice* principle (Section 3.1), and derived an analytical model for its FPR. Our analysis showed, however, that for some configurations, the Blocked BF still has a lower FPR compared to the Power-of-Two Blocked BF; therefore, we introduced the Power-of-1 + α Blocked BF to blend the advantages of both the Blocked and Power-of-Two Blocked schemes (Section 3.2). Using the parameter α, the Power-of-1 + α Blocked BF approach allows a fine-grained tuning of the algorithm for an optimal FPR.

Using the analytical model we presented for the Power-of-1 + α Blocked BF, we analyzed the FPR's dependence on α and depicted it in Figure 4. We found that it can be approximated using a linear function; however, we leave deeper analysis of this dependency to future work.

Acknowledgments

The authors would like to thank the anonymous reviewers whose comments helped to significantly improve the paper. Would also like to thank Isaac Keslassy for fruitful discussions at the early stages of this research and Mehmet Basoglu for his help with preparing this manuscript for publication. Finally, we thank Intel corp. for providing equipment and funds in support of this research.

References

[1] Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM **13** (1970) 422–426
[2] Broder, A., Mitzenmacher, M.: Network applications of bloom filters: A survey. Internet Mathematics **1** (2004) 485–509
[3] Fan, L., Cao, P., Almeida, J., Broder, A.: Summary cache: A scalable wide-area web cache sharing protocol. IEEE/ACM Transactions on Networking (TON) **8** (2000) 281–293
[4] Dharmapurikar, S., Krishnamurthy, P., Sproull, T., Lockwood, J.: Deep packet inspection using parallel bloom filters. In: High Performance Interconnects, 2003. Proceedings. 11th Symposium on. (2003) 44–51
[5] Hennessy, J., Patterson, D.: Computer architecture: a quantitative approach. Morgan Kaufmann (2003)
[6] Chen, Y., Kumar, A., Xu, J.: A New Design of Bloom Filter for Packet Inspection Speedup. In: IEEE Global Telecommunications Conference, 2007. GLOBECOM'07. (2007) 1–5
[7] Putze, F., Sanders, P., Singler, J.: Cache-, Hash-and Space-Efficient Bloom Filters. Lecture Notes in Computer Science **4525** (2007) 108
[8] Lumetta, S., Mitzenmacher, M.: Using the power of two choices to improve Bloom filters. Internet Mathematics **4** (2007) 17–33
[9] Mitzenmacher, M., Upfal, E.: Probability and computing: Randomized algorithms and probabilistic analysis. Cambridge University Press (2005)

Appendix

Lemma 1. *Let k be the number of hash functions used in a (Classic) BF and b be the number of blocks that the memory array consists of. Then, the average query operation will require accesses to $k - \mathcal{O}(\frac{1}{b})$ different blocks.*

Proof. Each query operation requires accesses to k different bits in the memory array. For a single bit look-up, the probability that a specific block is accessed is $\frac{1}{b}$. Conversely, the probability that a block is not accessed is $1 - \frac{1}{b}$. Therefore, the probability that a block is not accessed for any lookups of the k bits is $\left(1 - \frac{1}{b}\right)^k$, and the probability that the block is accessed by at least a single bit lookup is $1 - \left(1 - \frac{1}{b}\right)^k$. Based on that, the average number of blocks accessed by a query operation is $b\left[1 - \left(1 - \frac{1}{b}\right)^k\right]$. Using the Binomial expansion, we derive the required average number of accessed blocks:

$$b \cdot \left[1 - \left(1 - \frac{1}{b}\right)^k\right] = b \cdot \left[1 - \left(1 - \frac{k}{b} + \mathcal{O}(\frac{1}{b^2})\right)\right] = k - \mathcal{O}(\frac{1}{b})$$

□

Lemma 2. *In the special case of a fully balanced distribution among the blocks of a Blocked BF containing a total of n elements in m bits of memory split into b blocks, let $D_{Blocked\_Balanced}(j)$ be the probability of having exactly j elements in a block. Then, $FPR_{Blocked\_Balanced}$ is equal to FPR_{Basic}.*

Proof. Since the blocks are balanced, the number of elements in each block is equal to $\frac{n}{b}$; therefore, $D_{Blocked\_Balanced}(j)$ is defined as:

$$D_{Blocked\_Balanced}(j) = \begin{cases} 0 & j \neq \frac{n}{b} \\ 1 & j = \frac{n}{b} \end{cases} \tag{9}$$

Utilizing Equation 9 in Equation 2 yields:

$$\begin{aligned} FPR_{Blocked\_Balanced}(B,k) &= \sum_{j=0}^{\infty} D_{Blocked\_Balanced}(j) \cdot FPR_{Basic}(B,j,k) \\ &= FPR_{Basic}(B, \frac{n}{b}, k) = FPR_{Basic}(m,n,k) \end{aligned}$$

□

Lemma 3. *Let $D_{Blocked}(j)$ be the probability of having exactly j elements in a block of a Blocked BF as defined by Equation 3. Then, $D_{Blocked}(j)$ solves the iterative Equation 10 below.*

$$\frac{\partial D_{Blocked}(x)}{\partial n} = \frac{1}{b}\left(D_{Blocked}(x-1) - D_{Blocked}(x)\right) \tag{10}$$

Proof. We prove by using Equation 3 in Equation 10. According to Equation 3:

$$D_{Blocked}(x) \approx Poisson(\frac{n}{b}) = \frac{\left(\frac{n}{b}\right)^x \cdot e^{-\frac{n}{b}}}{x!}$$

$$\begin{aligned}\frac{\partial D_{Blocked}(x)}{\partial n} &= \frac{\partial}{\partial n}\left(\frac{\left(\frac{n}{b}\right)^x \cdot e^{-\frac{n}{b}}}{x!}\right) = \frac{\frac{x \cdot n^{x-1}}{b^x} \cdot e^{-\frac{n}{b}}}{x!} + \frac{\left(\frac{n}{b}\right)^x \cdot e^{-\frac{n}{b}}}{x!} \cdot \left(-\frac{1}{b}\right) \\ &= \frac{1}{b} \cdot \left(\frac{\left(\frac{n}{b}\right)^{x-1} \cdot e^{-\frac{n}{b}}}{(x-1)!} - \frac{\left(\frac{n}{b}\right)^x \cdot e^{-\frac{n}{b}}}{x!}\right) = \frac{1}{b}\left(D_{Blocked}(x-1) - D_{Blocked}(x)\right)\end{aligned}$$

Another way to prove it is by constructing a similar derivation to the iterative method used in Equation 6. □