Chapter 34
Efficient Update Control of Bloom Filter Replicas in Distributed Systems

Yifeng Zhu
University of Maine, USA

Hong Jiang
University of Nebraska – Lincoln, USA

ABSTRACT

This chapter discusses the false rates of Bloom filters in a distributed environment. A Bloom filter (BF) is a space-efficient data structure to support probabilistic membership query. In distributed systems, a Bloom filter is often used to summarize local services or objects and this Bloom filter is replicated to remote hosts. This allows remote hosts to perform fast membership query without contacting the original host. However, when the services or objects are changed, the remote Bloom replica may become stale. This chapter analyzes the impact of staleness on the false positive and false negative for membership queries on a Bloom filter replica. An efficient update control mechanism is then proposed based on the analytical results to minimize the updating overhead. This chapter validates the analytical models and the update control mechanism through simulation experiments.

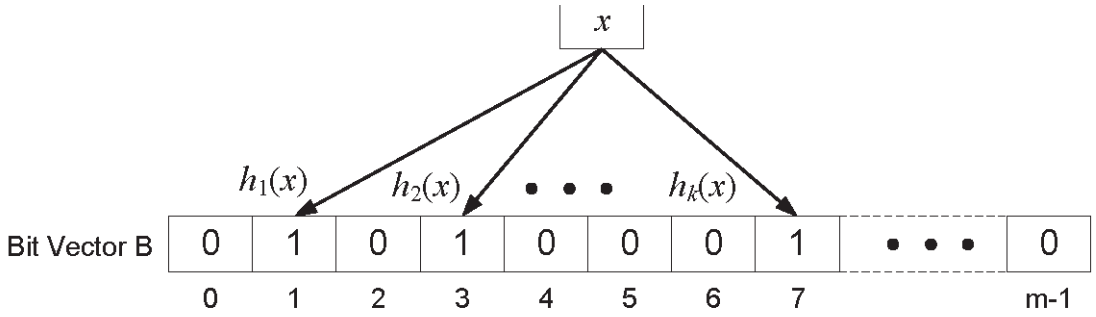
1. INTRODUCTION TO BLOOM FILTERS

A standard Bloom filter (BF) (Bloom, 1970) is a lossy but space-efficient data structure to support membership queries within a constant delay. As shown in Figure 1, a BF includes k independent random hash functions and a vector B of a length of m bits. It is assumed that the BF represents a finite set $S = \{x_1, x_2, \ldots, x_n\}$ of n elements from a universe U. The hash functions $h_i(x)$, $1 \le i \le k$, map the universe U to the bit address space $[1,m]$, shown as follows,

$$H(x) = \{h_i(x) \mid 1 \le h_i(x) \le m \text{ for } 1 \le i \le k\} \quad (1)$$

DOI: 10.4018/978-1-60566-661-7.ch034

Figure 1. A Bloom filter with a bit vector of m bits, and k independent hash functions. When an element x is added into the set represented, all bits indexed by those hash functions are set to 1.



Definition 1. For all $x \in U$, $B[H(x)] \equiv \{B[h_i(x)] \mid 1 \le i \le k\}$.

This notation facilitates the description of operations on the subset of B addressed by the hash functions. For example, $B[H(x)] = 1$ represents the condition in which all the bits in B at the positions of $h_1(x),\ldots$, and $h_k(x)$ are 1. "Setting $B[H(x)]$" means that the bits at these positions in B are set to 1.

Representing the set S using a BF B is fast and simple. Initially, all the bits in B are set to 0. Then for each $x \in S$, an operation of setting $B[H(x)]$ is performed. Given an element x, to check whether x is in S, one only needs to test whether $B[H(x)] = 1$. If no, then x is not a member of S; If yes, x is *conjectured* to be in S. Figure 1 shows the results after the element x is inserted into the Bloom filter.

A standard BF has two well-known properties that are described by the following two theorems.

Theorem 1.*Zero false negative*

For $\forall x \in U$, if $\exists i$, $B[h_i(x)] \neq 1$, then $x \notin S$.

For a static set S whose elements are not dynamically deleted, the bit vector indexed by those hash functions always never returns a false negative. The proof is easy and is not given in this chapter.

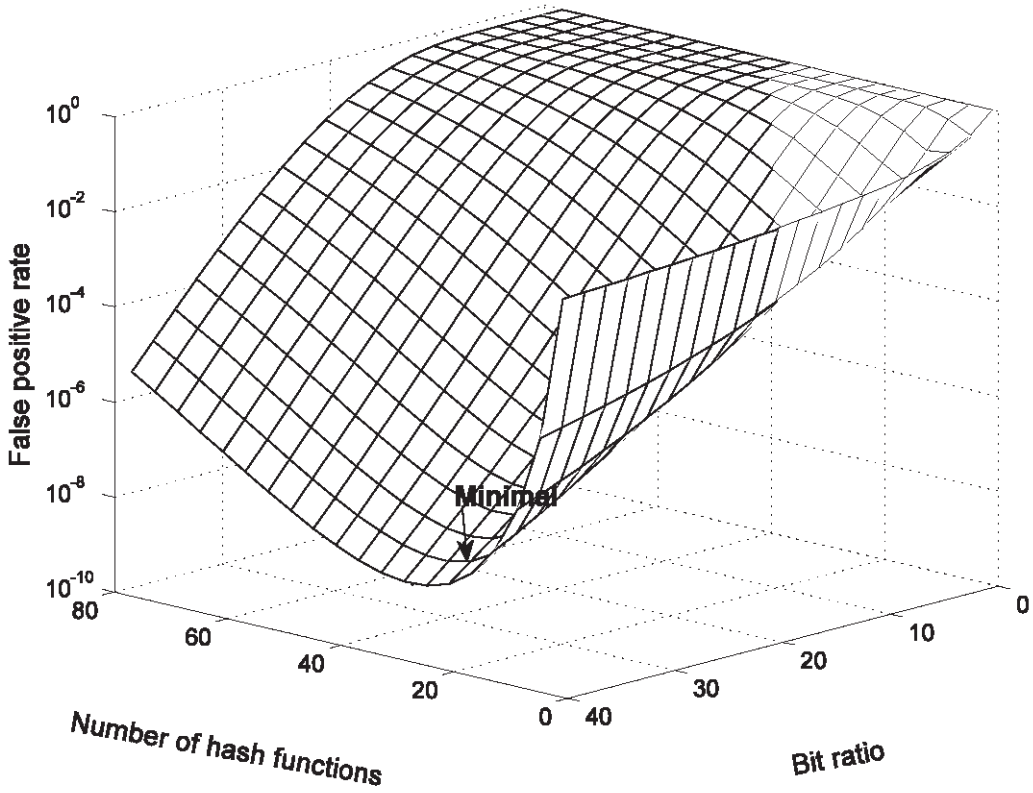
Theorem 2.*Possible false positive*

For $\forall x \in U$, if $B[H(x)] = 1$, then there is a small probability f^+ that $x \notin S$. This probability is called the *false positive rate* and $f^+ \approx (1 - e^{-kn/m})^k$. Given a specific ratio of m/n, f^+ is minimized when $k = (m/n)\ln 2$ and $f^+_{min} \approx (0.6185)^{m/n}$.

Proof: The proof is based on the mathematical model proposed in (James, 1983; McIlroy, 1982). Detailed proof can be found in (Li et al., 2000; Michael, 2002). For the convenience of the reader, the proof is briefly presented here.

After inserting n elements into a BF, the probability that a bit is zero is given by:

Figure 2. Expected false positive rate in a standard Bloom filter. A false positive is due to the collision of hash functions, where all indexed bits happen to be set by other elements.



$$P_0(n) = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m}. \tag{2}$$

Thus the probability that k bits are set to 1 is

$$P(k \text{ bits set}) = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx (1 - e^{-kn/m})^k. \tag{3}$$

Assuming each element is equally likely to be accessed and $|S||U|$, then the false positive rate is

$$f^+ = \left(1 - \frac{|S|}{|U|}\right) P(k \text{ bits set}) \approx (1 - e^{-kn/m})^k. \tag{4}$$

Given a specific ratio of $\frac{m}{n}$, i.e., the number of bits per element, it can be proved that the false posi-

tive rate f^+ is minimized when $k = \frac{m}{n} ln2$ and the minimal false positive rate is, as has been shown (Michael, 2002)

$$f^+ \approx 0.5^k = (0.6185)^{m/n} \quad (5)$$

The key advantage of a Bloom filter is that its storage requirement falls several orders of magnitude below the lower bounds of error-free encoding structures. This space efficiency is achieved at the cost of allowing a certain (typically none-zero) probability of *false positives*, that is, it may incorrectly return an "yes" although x is actually not in S. Appropriately adjusting m and k can minimize this probability of false-positive to a sufficiently small value so that benefits from the space and time efficiency far outweigh the penalty incurred by false positives in many applications. For example, when the bit-element ratio is 8 and the number of hash functions is 6, the expected false positive rate is only 0.0216. Figure 2 shows the false positive rate under different configurations.

In order to represent a dynamic set that is changing over time, (Li et al., 2000) proposes a variant named counting BF. A counting BF includes an array in which each entry is not a bit but rather a counter consisted of several bits. Counting Bloom filters can support element deletion operations. Let $C = \{c_j \mid 1 \le j \le m\}$ denote the counter vector and the counter c_j represents the difference between the number of settings and the number of unsetting operations made to the bit $B[j]$. All counters c_j for $1 \le j \le m$ are initialized to zero. When an element x is inserted or deleted, the counters $C[H(x)]$ are incremented or decreased by one, respectively. If c_j changes its value from one to zero, $B[j]$ is reset to zero. While this counter array consumes some memory space, (Li *et al*., 2000) shows that 4 bits per counter will guarantee the probability of overflow minuscule even with several hundred million elements in a BF.

2. APPLICATIONS OF BLOOM FILTERS IN DISTRIBUTED SYSTEMS

Bloom filters have been extensively used in many distributed systems where information dispersed across the entire system needs to be shared. For example, to reduce the message traffic, (Li et al, 2000) propose a web cache sharing protocol that employs a BF to represent the content of a web proxy cache and then periodically propagates that filter to other proxies. If a cache miss occurs on a proxy, that proxy checks the BFs replicated from other proxies to see whether they have the desired web objects in their caches. (Hong & Tao, 2003; Hua et al., 2008; Ledlie et al., 2002; Matei & Ian, 2002; Zhu et al., 2004; Zhu et al., 2008) use BFs to implement the function of mapping logical data identities to their physical locations in distributed storage systems. In these schemes, each storage node constructs a Bloom filter that summarizes the identities of data stored locally and broadcasts the Bloom filter to other nodes. By checking all filters collected locally, a node can locate the requested data without sending massive query messages to other nodes. Similar deployments of BFs have been found in geographic routing in wireless mobile systems (Pai-Hsiang, 2001), peer-to-peer systems (Hailong & Jun, 2004; John et al., 2000; Mohan & Kalogeraki, 2003; Rhea & Kubiatowicz, 2002), naming services (Little et al., 2002), and wireless sensor networks (Ghose et al. 2003; Luk et al. 2007).

A common characteristic of distributed applications of BFs, including all those described above, is that *a BF at a local host is replicated to other remote hosts to efficiently support distributed queries*. In such dynamical distributed applications, the information that a BF represents evolves over time. However, the

updating processes are usually delayed due to the network latency or the delay necessary in aggregating small changes into single updating message in order to reduce the updating overhead. Accordingly the contents of the remote replicas may become partially outdated. This possible staleness in the remote replicas not only changes the probability of false positive answers to membership queries on the remote hosts, but also brings forth the possibility of *false negatives*. A false negative occurs when a BF replica answers "no" to the membership query for an element while that element actually exists in its host. It is generated when a new element is added to a host while the changes of the BF of this host, including the addition of this new element, have not been propagated to its replicas on other hosts. In addition, this staleness also changes the probability of *false positives*, an event in which an element is incorrectly identified as a member. Throughout the rest of this chapter, the probabilities of false negatives and false positives are referred to as the false negative rate and false positive rate, respectively.

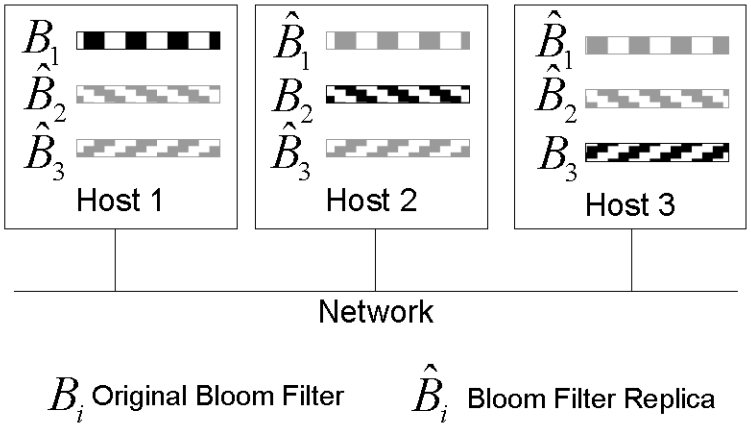
While the false negative and false positive rates for a BF at a local host have been well studied in the context of non-replicated BF (Bloom, 1970; Broder & Mitzenmacher, 2003; James, 1983; Li et al., 2000; Michael, 2002), very little attention has been paid to the false rates in the Bloom filter replicas in a distributed environment. In the distributed systems considered in this chapter, the false rates of the replicas are more important since most membership queries are performed on these replicas. A good understanding of the impact of the false negatives and false positives can provide the system designers with important and useful insights into the development and deployment of distributed BFs in such important applications as distributed file, database, and web server management systems in super-scales. Therefore, the first objective of this chapter is to analyze the false rates by developing analytical models and considering the staleness.

Since different application may desire a different tradeoff between false rate (e.g, miss/fault penalty) and update overhead (e.g., network traffic and processing due to broadcasting of updates), it is very important and significant for the systems overall performance to be able to control such a tradeoff for a given application adaptively and efficiently. The second objective is to develop an adaptive control algorithm that can accurately and efficiently maintain a desirable level of false rate for any given application by dynamically and judiciously adjusting the update frequency.

The primary contribution of this chapter is its developments of accurate closed-form expressions for the false negative and false positive rates in BF replicas, and the development of an adaptive replica-update control, based on our analytical model, that accurately and efficiently maintains a desirable level of false rate for any given application. To the best of our knowledge, this study is the first of its kind that has considered the impact of staleness of replicated BF contents in a distributed environment, and developed a mechanism to adaptively minimize such an impact so as to optimize systems performance.

The rest of the chapter is organized as follows. Section 3 presents our analytical models that theoretically derive false negative and false positive rates of a BF replica, as well as the overall false rates in distributed systems. Section 4 validates our theoretical results by comparing them against results obtained from extensive experiments. The adaptive updating protocols based on our theoretical analysis models are presented in Section 5. Section 6 gives related work and Section 7 concludes the chapter. The chapter is extended from our previous publication (Zhu & Jiang, 2006).

Figure 3. An example application of Bloom filters in a distributed system with 3 hosts.



3. FALSE RATES IN THEORY

In many distributed systems, the information about what data objects can be accessed through a host or where data objects are located usually needs to be shared to facilitate the lookup. To provide high scalability, this information sharing usually takes a decentralized approach, to avoid potential performance bottleneck and vulnerability of a centralized architecture such as a dedicated server. While BFs were initially used in non-distributed systems to save the memory space in the 1980's when memory was considered a precious resource (Lee, 1982; McIlroy, 1982), they have recently been extensively used in many distributed systems as a scalable and efficient scheme for information sharing, due to their low network traffic overhead.

The inherent nature of such information sharing in almost all these distributed systems, if not all, can be abstracted as a location identification, or mapping problem, which is described next. Without loss of generality, the distributed system considered throughout this chapter is assumed to consist of a collection of γ autonomous data-storing host computers dispersed across a communication network. These hosts partition a universe U of data objects into γ subsets $S_1, S_2,\ldots,S_\gamma$, with each subset stored on one of these hosts. Given an arbitrary object x in U, the problem is how to efficiently identify the host that stores x from any one of the hosts.

BFs are useful to solve this kind of problems. In a typical approach, each host constructs a BF representing the subset of objects stored in it, and then broadcasts that filter to all the other hosts. Thus each host keeps $\gamma - 1$ additional BFs, one for every other host. Figure 3 shows an example of a system with three hosts. Note that a filter $\hat{B}_i$ is a replica of B_i from Host i and $\hat{B}_i$ may become outdated if the changes to B_i are not propagated instantaneously. While the solution to the above information sharing problem can implemented somewhat differently giving rise to a number of solution variants (Hua et al., 2008; Ledlie et al., 2002; Zhu et al., 2004), the analysis of false rates presented in this chapter can be easily applied to these variants.

The detailed procedures of the operations of insertion, deletion and query of data objects are shown in Figure 4. When an object x is deleted from or inserted into Host i, the values of the counting filters $C_i[H(x)]$ and bits $B_i[H(x)]$ are adjusted accordingly. When the fraction of modified bits in B_i exceeds

Figure 4. Procedures of adding, deleting and querying object x at host i

AddObject(Object x, Host i)

1. Set $(B_i[H(x)])$ to 1;
2. Increase $C_i[H(x)]$ by 1;
3. **if**(the changed portion of B_i is larger than some threshold)
4. Multicast B_i to the other hosts;

DeleteObject(Object x, Host i)

1. Decrease $C_i[H(x)]$ by 1;
2. **for**($j=1$; $j \le k$; $j++$)
3. **if**($C_i[h_j(x)]=0$)
4. Unset bit $B_i[h_j(x)]$ to 0;

QueryObject(Object x, Host i)

1. $\psi = \varnothing$;
2. /* check the BF of the local host */
3. **if**($B_i[H(x)]=1$)
4. $\psi = \{i\}$;
5. /* check all BF replicas */
6. **for**($j=1$; $j \le \gamma$; $j++$)
7. **if**($j \neq i$ and $\hat{B}_i[H(x)]=1$)
8. $\psi = \psi \cup \{j\}$
9. **return** ψ;

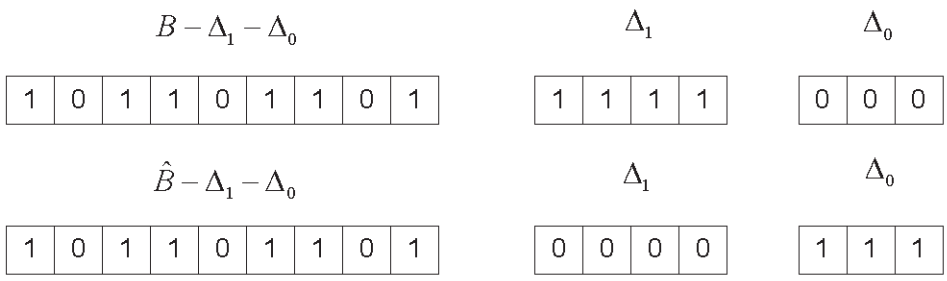
some threshold, B_i is broadcast to all the other hosts to update $\hat{B}_i$. To look up x, Host i performs the membership tests on all the BFs kept locally. If a test on B_i is positive, then x can potentially be accessed locally. If a test in the filter $\hat{B}_j$ for any $j \neq i$ is positive, then x is conjectured to be on Host j with high probability. Finally, if none of the tests is positive, x is considered nonexistent in the system.

In the following, we begin the analysis by examining the false negative and false positive rate of a single BF replica and then present the analysis of the overall false rates of all BFs kept locally on a host. The experimental validations of the analytical models are presented in the next section.

3.1. False Rates of Bloom Filter Replicas

Let B be a BF with m bits and $\hat{B}$ a replica of B. Let n and $\hat{n}$ be the number of objects in the set represented by B and by $\hat{B}$, respectively. We denote Δ_1 (Δ_0) as the set of all one (zero) bits in B that are

Figure 5. An example of a BF B and its replica $\hat{B}$ *where bits are reordered such that bits in* Δ_1 *and* Δ_0 *are placed together.*



different than (i.e., complement of) the corresponding bits in $\hat{B}$. More specifically,

$$\Delta_1 = \{B[i] \mid B[i] = 1, \hat{B}[i] = 0, \forall i \in [1, m]\}$$

$$\Delta_0 = \{B[i] \mid B[i] = 0, \hat{B}[i] = 1, \forall i \in [1, m]\}.$$

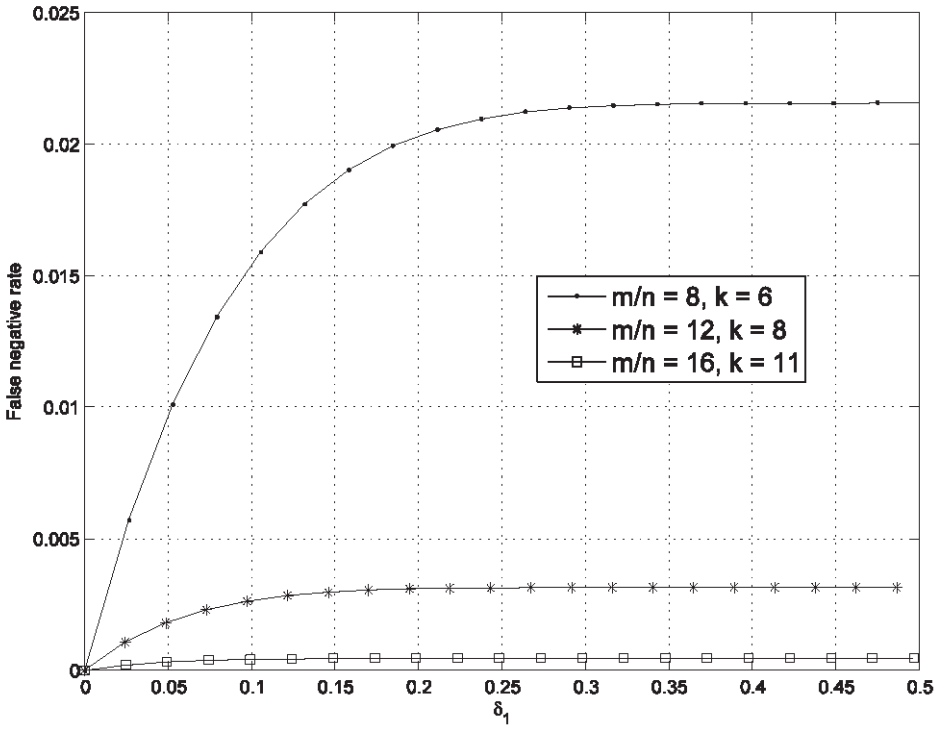
Thus, $\Delta_0 + \Delta_1$ represent the set of changed bits in *B* that have not been propagated to $\hat{B}$. The number of bits in this set is affected by the update threshold and update latency. Furthermore, if a nonempty Δ_1 is hit by least one hash function of a membership test on $\hat{B}$ while all other hash functions of the same test hit bits in $\hat{B} - \Delta_1 - \Delta_0$ with a value of one, then a false negative occurs in $\hat{B}$. Similarly, a false positive occurs if the nonempty Δ_1 is replaced by a nonempty Δ_0 in the exact membership test scenario on a $\hat{B}$ described above.

Lemma 1. Suppose that the numbers of bits in Δ_1 and in Δ_0 are $m\delta_1$ and $m\delta_0$, respectively. Then $\hat{n}$ is a random variable following a normal distribution with an extremely small variance (i.e., extremely highly concentrated around its mean), that is,

$$E(\hat{n}) = -\frac{m}{k}\ln(e^{-kn/m} + \delta_1 - \delta_0). \tag{6}$$

Proof: In a given BF representing a set of *n* objects, each bit is zero with probability $P_0(n)$, given in Equation 2, or one with probability $P_1(n) = 1 - P_0(n)$. Thus the average fractions of zero and one bits are $P_0(n)$ and $P_1(n)$, respectively. Ref. (Michael, 2002) shows formally that the fractions of zero and one bits are random variables that are highly concentrated on $P_0(n)$ and $P_1(n)$ respectively.

Figure 6. Expected false negative rate of a Bloom filter replica when the configuration of its original Bloom filter is optimal.



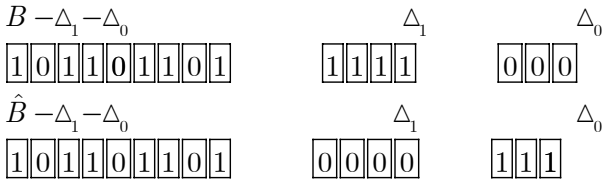


Figure 5 shows an example of B and $\hat{B}$ where bits in Δ_1 and Δ_0 are extracted out and placed together. The expected numbers of zero bits in $B - \Delta_1 - \Delta_0$ and in $\hat{B} - \Delta_1 - \Delta_0$ should be equal since the bits in them are always identical for any given B and $\hat{B}$. Thus for any given n, δ_1 and δ_0, we have

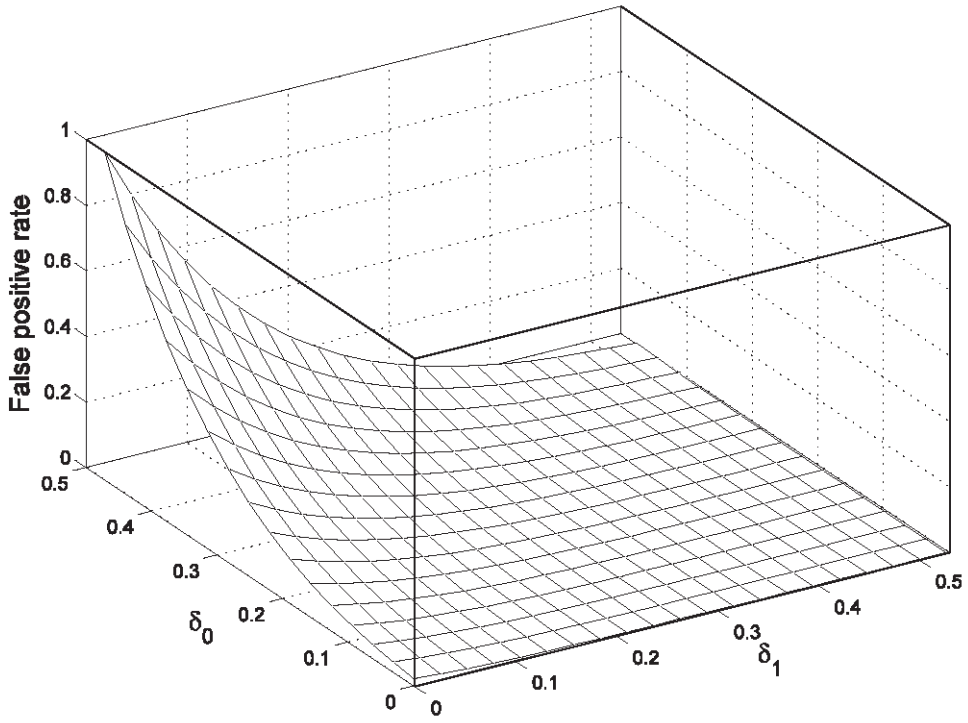
$$P_0(n) - \delta_0 = E(P_0(\hat{n})) - \delta_1 \tag{7}$$

Substituting Equation 2 into the above equation, we have

$$e^{-kn/m} - \delta_0 = e^{-kE(\hat{n})/m} - \delta_1 \tag{8}$$

After solving Equation 8, we obtain Equation 6.

Figure 7. Expected false positive rate of a Bloom filter replica when the configuration of its original Bloom filter is optimal.



Pragmatically, in any given BF with n objects, the values of δ_1 and δ_0, which represent the probabilities of a bit falling in Δ_1 and Δ_0 respectively, are relatively small. Theoretically, the number of bits in Δ_1 is less than the total number of one bits in B, thus we have $\delta_1 \leq 1 - e^{-kn/m}$. In a similar way, we can conclude that $\delta_0 \leq e^{-kn/m}$.

Theorem 3.*False Negative Rate*

The expected false negative rate $\hat{f}^-$ in the BF replica $\hat{B}$ is $P_1(n)^k - (P_1(n) - \delta_1)^k$, where $P_1(n) = 1 - e^{-kn/m}$.

Proof: As mentioned earlier, a false negative in $\hat{B}$ occurs when at least one hash function hits the bits in Δ_1 in $\hat{B}$ while the others hit the bits in $\hat{B} - \Delta_1 - \Delta_0$ with a value of one. Hence, the false negative rate is

$$\hat{f}^- = \sum_{i=1}^{k} \binom{k}{i} \delta_1^i \left(P_1(\hat{n}) - \delta_0\right)^{k-i} = \left(P_1(\hat{n}) - \delta_0 + \delta_1\right)^k - \left(P_1(\hat{n}) - \delta_0\right)^k$$

Since $P_0(n) = 1 - P_1(n)$ and $P_0(\hat{n}) = 1 - P_1(\hat{n})$, Equation 7 can be rewritten as:

$$E(P_1(\hat{n})) = P_1(n) + \delta_0 - \delta_1 \tag{9}$$

Hence

$$E(\hat{f}^-) = \left(E(P_1(\hat{n})) - \delta_0 + \delta_1\right)^k - \left(E(P_1(\hat{n})) - \delta_0\right)^k$$

$$= P_1(n)^k - \left(P_1(n) - \delta_1\right)^k \tag{10}$$

Figure 6 shows the expected false negative rate when the false positive of the original BF is minimized. The minimal false positive rate is 0.0214, 0.0031 and 0.00046 when the bit-element ratio is 8, 12 and 16 respectively. Figure 6 shows that the false negative rates of a BF replica are more than 50% of the false positive rates of the original BF when δ_1 is 5%, and more than 75% when δ_1 is 10%. This proves that the false negative may be significant and should not be neglected in distributed applications.

Theorem 4.*False Positive Rate*

The expected false positive rate $\hat{f}^+$ for the Bloom filter replica $\hat{B}$ is $(P_1(n) + \delta_0 - \delta_1)^k$, where $P_1(n) = 1 - e^{-kn/m}$.

Proof: If $\hat{B}$ confirms positively the membership of an object while this object actually does not belong to *B*, then a false positive occurs. More specifically, a false positive occurs in $\hat{B}$ if for any $x \notin B$, all hit bits by hash functions of the membership test for *x* are ones in $\hat{B} - \Delta_1 - \Delta_0$, or for any $x \in U$, all hit bits are ones in $\hat{B}$ but at least one hit bit is in Δ_0. Thus, we find that

$$\hat{f}^+ = \left(1 - \frac{n}{|U|}\right)\left(P_1(\hat{n}) - \delta_0\right)^k + \sum_{i=1}^{k} \binom{k}{i} \delta_0^i (P_1(\hat{n}) - \delta_0)^{k-i}$$

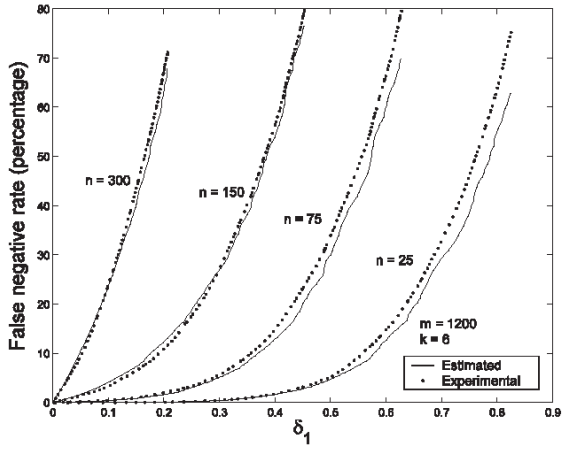
$$= P_1(\hat{n})^k - \frac{n}{|U|}(P_1(\hat{n}) - \delta_0)^k \tag{11}$$

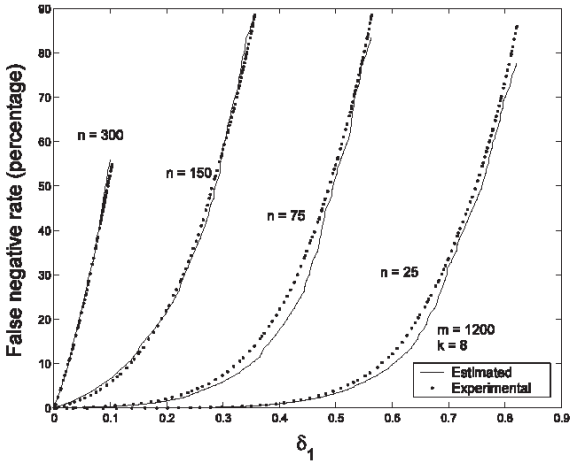
Considering $n \,|U$ and Equation 9, we have

$$E(\hat{f}^+) = (E(P_1(\hat{n})))^k - \frac{n}{|U|}(E(P_1(\hat{n})) - \delta_0)^k$$

$$= \left(P_1(n) + \delta_0 - \delta_1\right)^k - \frac{n}{|U|}(P_1(n) - \delta_1)^k$$

Figure 8. Comparisons of estimated and experimental $\hat{f}^{-}$ of $\hat{B}$ when k is 6, 8 and 11 respectively. The initial object number in both B and $\hat{B}$ is 25, 75, 150 and 300 (m = 1200).





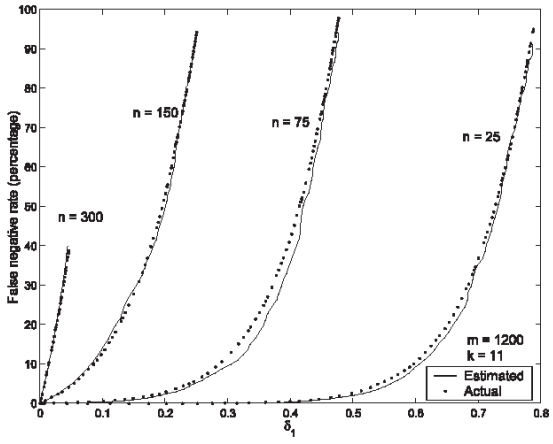


Table 1. False positive rates comparisons when k is 6 and 8 respectively (m = 1200).

				$\hat{f}^{+}$ **(percentage)**	
k	$\hat{n}$	δ_0	δ_1	Estimated	Experimental
6	25	0.0942	0.2042	0.0002	0
6	25	0.0800	0.3650	0.0002	0
6	25	0.0600	0.4875	0.0001	0
6	75	0.0800	0.1608	0.0934	0.1090
6	75	0.0600	0.2833	0.0794	0.1090
6	75	0.0483	0.3758	0.0799	0.1090
6	150	0.0533	0.1042	2.2749	2.6510
6	150	0.0400	0.1800	2.3540	2.6510
6	150	0.0325	0.2508	2.1872	2.6530
6	300	0.0250	0.0417	23.6555	25.4790
6	300	0.0183	0.0692	25.4016	25.4710
6	300	0.0117	0.1000	24.7241	25.4750
8	25	0.1083	0.2425	0.00002	0
8	25	0.0792	0.4192	0.00002	0
8	25	0.0550	0.5425	0.00002	0
8	75	0.0792	0.1767	0.0525	0.0540
8	75	0.0550	0.3000	0.0504	0.0540
8	75	0.0425	0.3917	0.0506	0.0540
8	150	0.0475	0.1050	2.5163	2.5770
8	150	0.0350	0.1758	2.6783	2.5780
8	150	0.0283	0.2367	2.5384	2.5790
8	300	0.0192	0.0333	33.2078	33.2580
8	300	0.0133	0.0558	34.4915	33.2550
8	300	0.0083	0.0817	32.1779	33.2550

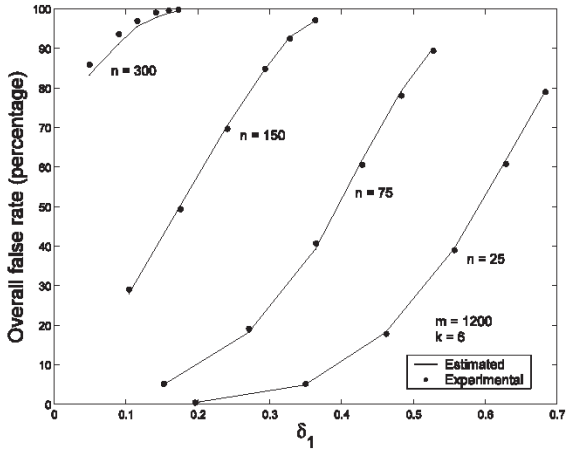
$$\approx \left(P_1(n) + \delta_0 - \delta_1\right)^k \tag{12}$$

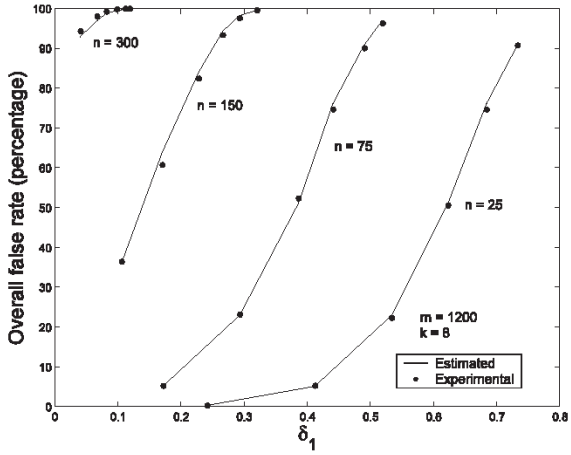
3.2. Overall False Rates

In the distributed system considered in this study, there are a total of γ hosts and each host has γ BFs, with γ−1 of them replicated from the other hosts. To look up an object, a host performs the membership tests in all the BFs kept locally. This section analyzes the overall false rates on each BF replica and each host.

Give any BF replica $\hat{B}$, the events of a false positive and a false negative are exclusive. Thus it is

Figure 9. Comparisons of estimated and experimental $f_{overall}$ in a distributed system with 5 hosts when k is 6, 8, and 11 respectively. The initial object number n on each host is 25, 75, 150 and 300 respectively. Then each host adds a set of new objects. The number of new objects on each host increases from 50 to 300 with a step size of 50. (m = 1200)





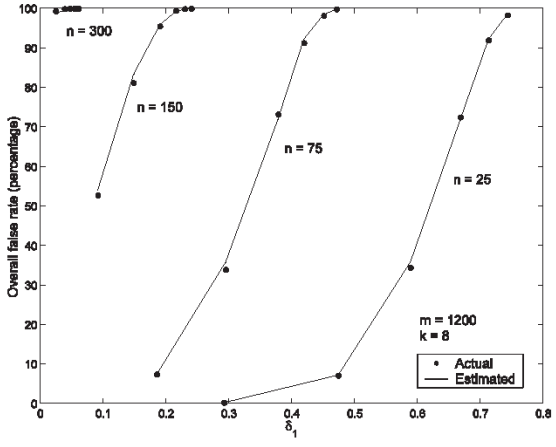


Table 2. Overall false rate comparisons under optimum initial operation state when k is 6 and 8 respectively. 100 new objects are added on each host and then a set of existing objects are deleted from each host. The number of deleted objects increases from 10 to 100 with a step size of 10. (m = 1200) In the first group, initially Initially n = 150 and m/n = 8; in the second group, n = 100 and m/n = 12 initially.

			$f_{overall}$ **(percentage)**	
k	δ_0	δ_1	Estimated	Experimental
6	0.0100	0.1705	46.2259	45.2200
6	0.0227	0.1657	42.4850	40.6880
6	0.0347	0.1627	38.7101	37.2420
6	0.0458	0.1582	34.9268	33.8460
6	0.0593	0.1545	31.3748	30.4540
6	0.0715	0.1497	27.8831	27.3700
6	0.0837	0.1445	24.5657	24.8000
6	0.0938	0.1392	21.2719	22.5560
6	0.1045	0.1340	18.2490	20.4520
6	0.1165	0.1300	15.5103	18.7540
8	0.0123	0.2375	30.9531	29.6280
8	0.0255	0.2275	25.7946	23.6280
8	0.0413	0.2180	21.0943	18.0000
8	0.0552	0.2123	16.7982	14.6720
8	0.0658	0.2043	12.9800	12.0040
8	0.0772	0.1965	9.7307	9.7320
8	0.0920	0.1900	7.1016	7.7520
8	0.1075	0.1848	4.9936	6.1280
8	0.1237	0.1788	3.4031	4.8400
8	0.1377	0.1732	2.2034	3.8160

easy to find that the overall false rate of $\hat{B}$ is

$$E(f_{overall}) = E(f^-) + E(f^+) \tag{13}$$

where $E(f^-)$ and $E(f^+)$ are given in Equation 10 and 12 respectively.

On Host i, BF B_i represents all the objects stored locally. While only false positives occur in B_i, both false positives and false negatives can occur in the replicas $\hat{B}_j$ for any $j \neq i$. Since the failed membership test in any BF leads to a lookup failure, the overall false positive and false negative rates on Host i are therefore

$$E(f_{host}^{+}) = 1 - (1 - f_i^{+}) \prod_{j=1, j \neq i}^{\gamma} (1 - \hat{f}_j^{+}) \tag{14}$$

and

$$E(f_{host}^{-}) = 1 - \prod_{j=1, j \neq i}^{\gamma} (1 - \hat{f}_j^{-}) \tag{15}$$

where f_i^{+}, $\hat{f}_j^{-}$ and $\hat{f}_j^{+}$ are given in Theorem 2, 3 and 4 respectively.

The probability that Host *i* fails a membership lookup can be expressed as follows:

$$E(f_{host}) = E(f_{host}^{+} + f_{host}^{-} - f_{host}^{+} f_{host}^{-}) \tag{16}$$

In practice, we can use the overall false rate of a BF replica to trigger updating process and use the overall false rate of all BFs on a host to evaluate the whole systems. In a typical distributed environment with many nodes, the updating of a Bloom filter replica $\hat{B}_i$ stored on node *j* can be triggered by either the home node *i* or the node *j*. Since many nodes hold the replica of B_i, it is more efficient to let the home node *i* to initiate the updating process of all replicas of B_i. Otherwise, the procedure of checking whether an updating is needed would be performed by all other nodes, wasting both network and CPU resources. Accordingly, we can only use the overall false rate of a BF replica $E(f_{overall})$ as the updating criteria. On the other hand, $E(f_{host})$ can be used to evaluate the overall efficiency of all BFs stored on the same host.

4. EXPERIMENTAL VALIDATION

This section validates our theoretical framework developed in this chapter by comparing the analytical results produced by our models with experimental results obtained through real experiments.

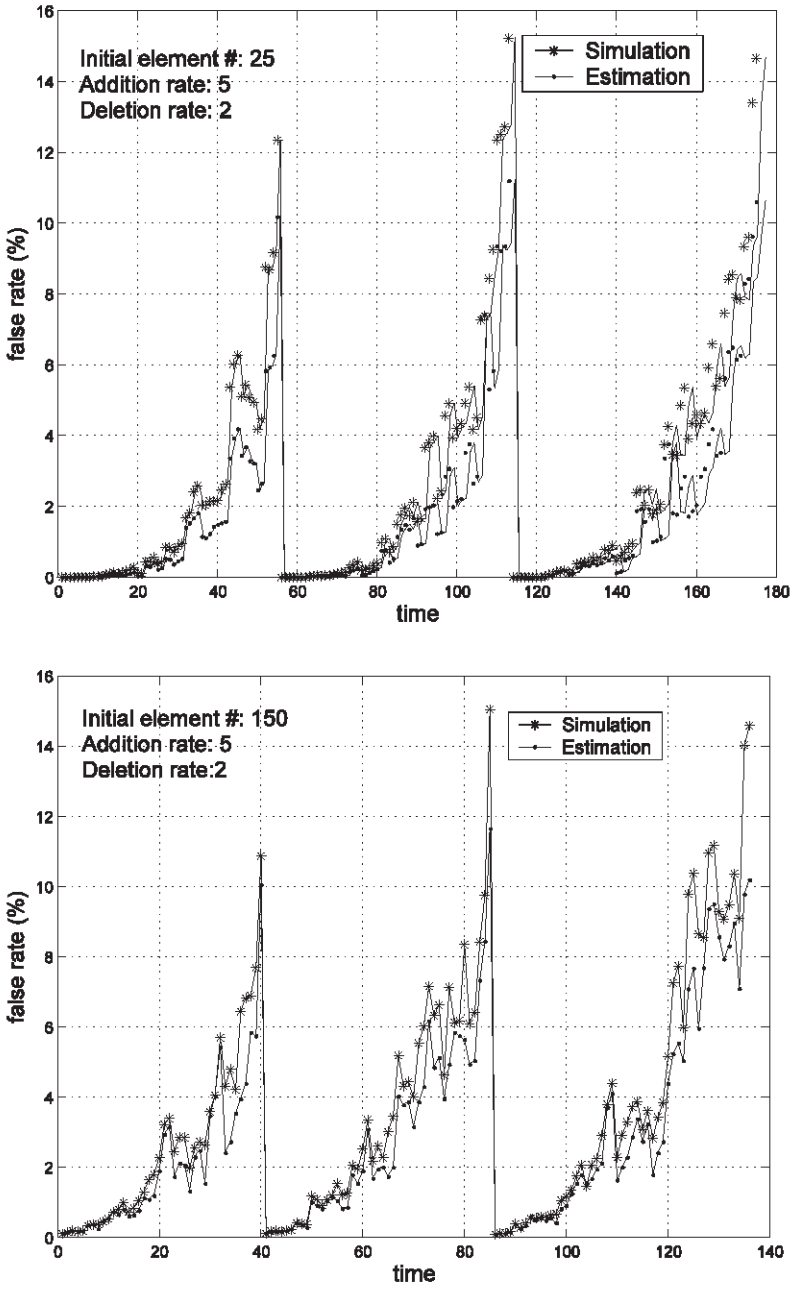
We begin by examining a single BF replica. Initially the Bloom filter replica $\hat{B}$ is exactly the same as *B*. Then we artificially change *B* by randomly inserting new objects into *B* or randomly deleting existing objects from *B* repeatedly. For each specific modification made to *B*, we calculate the corresponding δ_1 and δ_0 and use 100,000 randomly generated objects to test the memberships against $\hat{B}$. Since the actual objects represented in *B* are known in the experiments, the false negative and positive rates can be easily measured.

Figure 8 compares analytical and real false negative rates, obtained from the theoretic models and from experiments respectively, by plotting the false negative rate in $\hat{B}$ as a function of δ_1, a measure of update threshold, for different numbers of hashing functions ($k = 6$ and $k = 8$) when the initial number of objects in *B* are 25, 75, 150 and 300 respectively. Since the false negative rates are independent of

δ_0, only object deletions are performed in B.

Table 1 compares the analytical and real false positive rates of $\hat{B}$ when k is 6 and 8 respectively. In

Figure 10. In an environment of two servers, the figures show the overall false rate on one server when the initial number of elements in one server are 25 and 150 respectively. The ratio of bits per element is 8 and 6 hash functions are used. The rate for element addition and deletion are respectively 5 and 2 per time unit on each server.



these experiments, both object deletions and additions are performed in B while $\hat{B}$ remains unaltered. It is interesting that the false positive rates of $\hat{B}$ is kept around some constant for a specific $\hat{n}$ although the objects in B changes in the real experiments. It is true that if the number of objects in B increases or decreases, the false positive rate in $\hat{B}$ should decrease or increase accordingly before the changes of B is propagated to $\hat{B}$. However, due to the fact that n is far less than the total object number in the universe U, the change of the false positive rate in $\hat{B}$ is too small to be perceptible. These tests are made accordant with the real scenarios of BF applications in distributed systems. In such real applications, the number of possible objects is usually very large and thus BFs are deployed to efficiently reduce the network and network communication requirements. Hence, in these experiments the number of objects used to test $\hat{B}$ is much larger than the number of objects in B or $\hat{B}$ (100,000 random objects are tested). Under such large size of testing samples, the influence of the modification in B on the false positive rate of $\hat{B}$ is difficult to be observed.

We also simulated the lookup problem in a distributed system with 5 hosts. Figure 9 shows the comparisons of the analytical and experimental average overall false rates on each host. In these experiments, we only added new objects without deleting any existing items so that δ_0 is kept zero. The experiments presented in Table 2 considers both the deletion and addition of objects on each host when the initial state of BF on each host is optimized, this is, the number of hash functions is the optimal under the ratio between m and the initial number of objects n. This specific setting aims to emulate the real application where m/n and k are usually optimally or sub-optimally matched by dynamically adjusting the BF length m (Hong & Tao, 2003) or designing the BF length according to the average number of objects (Ledlie et al., 2002; Li et al., 2000; Little et al., 2002; Matei & Ian, 2002; Zhu et al., 2004). All the analytical results have been very closely matched by their real (experimental) counterparts consistently, strongly validating our theoretical models.

5. REPLICA UPDATE PROTOCOL

To reduce the false rate caused by staleness, the remote Bloom filter replica needs to be periodically updated. An update process is typically triggered if the percentage of dirty bits in a local BF exceeds some threshold. While a small threshold causes large network traffic and a large threshold increases the false rate, this tradeoff is usually reached by a trial-and-error approach that runs numerous (typically a large number of) trials in real experiments or simulations. For example, in the summery cache study (Li et al., 2000), it is recommended that if 10 percent of bits in a BF are dirty, then the BF propagates its changes to all replicas. However, this approach has the following disadvantages.

1. It cannot directly control the false rate. To keep the false rate under some target value, complicated simulations or experiments have to be conducted to adjust the threshold for dirty bits. If the target false rate changes, this tedious process has to be repeated to find a "golden" threshold.
2. It treats all dirty bits equally and does not distinguish the zero-dirty bits from the one-dirty bits. In fact, as shown in previous sections, the dirty one bits and the dirty zero bits exert different impacts on the false rates.

3. It does not allow flexible update control. In many applications, the penalty of a false positive and a false negative are significantly different. For example, in summery cache (Li et al., 2000), a false positive occurs if a request is not a cache hit on some web proxy when the corresponding Bloom filter replica confirms so. The penalty of a false positive is a waste of query message to this local web proxy. A false negative happens if a request can be hit in a local web proxy but the Bloom filter replica mistakenly indicates otherwise. The penalty of a false negative is a round-trip delay in retrieving information from a remote web server through the Internet. Thus, the penalty of a false negative is much larger than that of a false positive. The updating protocols based on the percentage of dirty bits do not allow one to place more weight on the false negative rate, thus limiting the flexibility and efficiency of the updating process.

Based on the theoretic models presented in the previous sections, an updating protocol that directly controls the false rate is designed in this chapter. In a distributed system with γ nodes where each node has a local BF to represent all local elements, each node is responsible for automatically updating its BF replicas. Each node estimates the false rate of its remote BF replica and if the false rate exceeds some desire false rate, as opposed to a predetermined threshold on the percentage of dirty bits in the conventional updating approaches, a updating process is triggered. To estimate the false rate of remote BF replica $\hat{B}$, each node has to record the number of elements stored locally (n), in addition to a copy of remote BF replica $\hat{B}$. This copy is essentially the local BF B when the last updating is made. It is used to calculate the percentage of dirty one bits (δ_1) and the dirty zero bits (δ_0). Compared with the conventional updating protocols based on the total percentage of dirty bits, this protocol only needs to record one more variable (n), thus it does not significantly increase the maintenance overhead.

This protocol allows more flexible updating protocols that consider the penalty difference between a false positive and a false negative. The overall false rate can be a weighted sum of the false positive rate and the false negative rate, shown as follows:

$$E(f_{overall}) = w^{+}E(f^{+}) + w^{-}E(f^{-}) \tag{17}$$

where w^{+} and w^{-} are the weights. The values of w^{+} and w^{-} depends on the applications and also the application environments.

We prove the effectiveness of this update protocol through event driven simulations. In this simulation, we made the following assumptions.

1. Each item is randomly accessed. This assumption may not be realistic in some real workloads, in which an item has a greater than equal chance of being accessed again once it has been accessed. Though all previous theoretic studies on Bloom filters assume a workload with uniform access spectrum, further studies are needed to investigate the impact of this assumption.
2. Each local node deletes or adds items at a constant rate. In fact, the deletion and addition rate changes dynamically throughout the lifetime of applications. This simplifying assumption is employed just to prove our concept while keeping our experiments manageable in the absence of a real trace or benchmark.

3. The values of w^+ and w^- are 1. Their optimal values depend on the nature of the applications and environments.

We simulate a distributed system with two nodes where each node keeps a BF replica of the other. We assume the addition and deletion are 5 and 2 per time unit respectively and our desired false rate is 10%. Figure 10 shows the estimated false rate and the measured false rate of node 1 throughout the deletion, addition and updating processes. Due to the space limitation, the false rate on node 2, which is similar to node 1, is not shown in this chapter. In addition, we have changed the addition rate and deletion rates. Simulation results consistently indicate that our protocol is accurate and effective in control the false rate.

6. RELATED WORK

Standard Bloom filters (Bloom, 1970) have inspired many extensions and variants, such as the Counting Bloom filters (Li et al., 2000), compressed Bloom filters (Michael, 2002), the space-code Bloom filters (Kumar et al., 2005), the spectral Bloom filters (Saar & Yossi, 2003), time-decaying Bloom filters (Cheng et al., 2005), and the Bloom filter state machine (Bonomi et al., 2006). The counting Bloom filters are used to support the deletion operation and handle a set that is changing over time (Li et al., 2000). Time-decaying Bloom filters maintains the frequency count for each item stored in the Bloom filters and the values of these frequency count decay with time (Cheng et al., 2005). Multi-Dimension Dynamic Bloom Filters (MDDBF) supports representation and membership queries based on the multi-attribute dimension (Guo et al., 2006). Its basic idea is to represent a dynamic set A with a dynamic $s \times m$ bit matrix, in which there are s standard Bloom filters and each Bloom filter has a length of m bits. A novel Parallel Bloom Filters (PBF) and an additional hash table has been developed to maintain multiple attributes of items and verify the dependency of multiple attributes, thereby significantly decreasing false positives (Hua & Xiao, 2006).

Bloom filters have significant advantages in space saving and fast query operations and thus have been widely applied in many distributed computer applications, such as aiding longest prefix matching (Dharmapurikar et al., 2006), and packet classification (Baboescu & Varghese, 2005). Extended Bloom filter provides better throughput performance for router applications based on hash tables by using a small amount of multi-port on-chip memory (Song et al., 2005). Whenever space is a concern, a Bloom filter can be an excellent alternative to storing a complete explicit list.

In many distributed applications, BFs are often replicated to multiple hosts to support membership query without contacting other hosts. However, these replicas might become stale since the changes of BFs usually cannot be propagated instantly to all replicas in order to reduce the update overhead. As a result, the BF replicas may return false negatives. This observation motivates the research presented in this chapter.

7. CONCLUSION

Although false negatives do not occur in standard BF, this chapter shows that the staleness in a BF replica can produce false negative. We presents the theoretical analysis of the impact of staleness existing in

many distributed BF applications on the false negative and false positive rates, and developed an adaptive update control mechanism that accurately and efficiently maintains a desirable level of false rate for a given application. To the best of our knowledge, we are the first to derive accurate closed-form expressions that incorporate the staleness into the analysis of the false negative and positive rates of a single BF replica, to develop the analytical models of the overall false rates of BF arrays that have been widely used in many distributed systems, and to develop an adaptively controlled update process that accurately maintains a desirable level of false rate for a given application. We have validated our analysis by conducting extensive experiments. The theoretical analysis presented not only provides system designers with significant theoretical insights into the development and deployment of BFs in distributed systems, but also are useful in practice for accurately determining when to trigger the processes of updating BF replicas in order to keep the false rates under some desired values, or, equivalently, minimize the frequency of updates to reduce update overhead.

7. ACKNOWLEDGMENT

This work was partially supported by a faculty startup grant of University of Maine, and National Science Foundation Research Grants (CCF #0621493, CCF #0754951, CNS #0723093, DRL #0737583, CNS #0619430).

REFERENCES

Baboescu, F., & Varghese, G. (2005). Scalable packet classification. *IEEE/ACM Trans. Netw.*, *13*(1), 2–14.

Bloom, H. B. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, *13*(7), 422–426. doi:10.1145/362686.362692

Bonomi, F., Mitzenmacher, M., Panigrah, R., Singh, S., & Varghese, G. (2006). *Beyond bloom filters: from approximate membership checks to approximate state machines*. Paper presented at the Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications.

Broder, A., & Mitzenmacher, M. (2003). *Network Applications of Bloom Filters: A Survey. Internet Mathematics*, *1*(4), 485–509.

Cheng, K., Xiang, L., Iwaihara, M., Xu, H., & Mohania, M. M. (2005). *Time-Decaying Bloom Filters for Data Streams with Skewed Distributions*. Paper presented at the Proceedings of the 15th International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications.

Dharmapurikar, S., Krishnamurthy, P., & Taylor, D. E. (2006). Longest prefix matching using bloom filters. *IEEE/ACM Trans. Netw.*, *14*(2), 397–409.

Ghose, F., Grossklags, J., & Chuang, J. (2003). *Resilient Data-Centric Storage in Wireless Ad-Hoc Sensor Networks.* Proceedings the 4th International Conference on Mobile Data Management (MDM'03), (pp. 45-62).

Guo, D., Wu, J., Chen, H., & Luo, X. (2006). *Theory and Network Applications of Dynamic Bloom Filters.* Paper presented at the INFOCOM 2006. 25th IEEE International Conference on Computer Communications.

Hailong, C., & Jun, W. (2004). *Foreseer: a novel, locality-aware peer-to-peer system architecture for keyword searches*. Paper presented at the Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware.

Hong, T., & Tao, Y. (2003). *An Efficient Data Location Protocol for Self.organizing Storage Clusters*. Paper presented at the Proceedings of the 2003 ACM/IEEE conference on Supercomputing.

Hua, Y., & Xiao, B. (2006). A Multi-attribute Data Structure with Parallel Bloom Filters for Network Services. *Proceedings of 13th International Conference of High Performance Computing (HiPC),*(pp. 277-288).

Hua, Y., Zhu, Y., Jiang, H., Feng, D., & Tian, L. (2008). Scalable and Adaptive Metadata Management in Ultra Large-Scale File Systems. *Proceedings of the 28th International Conference on Distributed Computing Systems (ICDCS 2008)*.

James, K. M. (1983). A second look at bloom filters. *Communications of the ACM*, *26*(8), 570–571. doi:10.1145/358161.358167

John, K., David, B., Yan, C., Steven, C., Patrick, E., & Dennis, G. (2000). OceanStore: an architecture for global-scale persistent storage. *SIGPLAN Not.*, *35*(11), 190–201. doi:10.1145/356989.357007

Kumar, A., Xu, J., & Zegura, E. W. (2005). *Efficient and scalable query routing for unstructured peer-to-peer networks.* Paper presented at the Proceedings INFOCOM 2005, 24th Annual Joint Conference of the IEEE Computer and Communications Societies.

Ledlie, J., Serban, L., & Toncheva, D. (2002). *Scaling Filename Queries in a Large-Scale Distributed File System*. Harvard University, Cambridge, MA.

Lee, L. G. (1982). Designing a Bloom filter for differential file access. *Communications of the ACM*, *25*(9), 600–604. doi:10.1145/358628.358632

Li, F., Pei, C., Jussara, A., & Andrei, Z. B. (2000). Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, *8*(3), 281–293.

Little, M. C., Shrivastava, S. K., & Speirs, N. A. (2002)... *The Computer Journal*, *45*(6), 645–652. doi:10.1093/comjnl/45.6.645

Luk, M., Mezzour, G., Perrig, A., & Gligor, V. (2007). MiniSec: A Secure Sensor Network Communication Architecture. *Proceedings of IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, (pp. 479-488).

Matei, R., & Ian, F. (2002). *A Decentralized, Adaptive Replica Location Mechanism*. Paper presented at the Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing.

McIlroy, M. (1982). Development of a Spelling List. *Communications, IEEE Transactions on [legacy, pre - 1988], 30*(1), 91-99.

Michael, M. (2002). Compressed bloom filters. *IEEE/ACM Trans. Netw.*, *10*(5), 604–612.

Mohan, A., & Kalogeraki, V. (2003). *Speculative routing and update propagation: a kundali centric approach.* Paper presented at the IEEE International Conference on Communications, 2003.

Pai-Hsiang, H. (2001). *Geographical region summary service for geographical routing*. Paper presented at the Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking and computing.

Rhea, S. C., & Kubiatowicz, J. (2002). *Probabilistic location and routing.* Paper presented at the IEEE INFOCOM 2002, Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies Proceedings.

Saar, C., & Yossi, M. (2003). *Spectral bloom filters*. Paper presented at the Proceedings of the 2003 ACM SIGMOD international conference on Management of data.

Song, H., Dharmapurikar, S., Turner, J., & Lockwood, J. (2005). *Fast hash table lookup using extended bloom filter: an aid to network processing*. Paper presented at the Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications.

Zhu, Y., & Jiang, H. (2006). *False Rate Analysis of Bloom Filter Replicas in Distributed Systems*. Paper presented at the Proceedings of the 2006 International Conference on Parallel Processing.

Zhu, Y., Jiang, H., & Wang, J. (2004). *Hierarchical Bloom filter arrays (HBA): a novel, scalable metadata management system for large cluster-based storage*. Paper presented at the Proceedings of the 2004 IEEE International Conference on Cluster Computing.

Zhu, Y., Jiang, H., Wang, J., & Xian, F. (2008). *HBA: Distributed Metadata Management for Large Cluster-Based Storage Systems*. *IEEE Transactions on Parallel and Distributed Systems*, *19*(6), 750–763. doi:10.1109/TPDS.2007.70788

KEY TERMS AND DEFINITIONS

Bloom Filter: A Bloom filter is a space-efficient data structure that supports membership queries. It consists of a bit array and all bits are initially set to 0. It uses a fix number of predefined independent hash functions. For each element, all hashed bits are set to 1. To check whether an element belongs to the set represented by a Bloom filter, one simply checks all bits pointed by the hash functions are 1. If not, the element is not in the set. If yes, the element is consider as a member.

Bloom Filter Array: A Bloom filter array, consisted of multiple Bloom filters, represents multiple sets. It is a space-efficient data structure to evaluate whether an element is within these sets and which set this element belongs to if yes.

Bloom Filter Replica: A Bloom filter replica is a replication of a Bloom filter. In a distributed environment, the original and replicated Bloom filters are typically stored on different servers for improved performance and fault tolerance. A Bloom filter replica will generate both false positives and false negatives.

Bloom Filter Update Protocol: When the set that a Bloom filter represents is changed over time, the corresponding Bloom filter replica becomes out-dated. In order to reduce the probability that the Bloom filter replica reports the membership incorrectly, the replica needs to be updated frequently. The Bloom filter update protocol determines when a Bloom filter replica needs to be updated.

Distributed Membership Query: Membership query is one fundamental function that reports where the target data, resource, or service is located. The membership query can be performed by a centralized server or by a group of distributed server. The latter approach has a stronger scalability and is referred as distributed memory query.

False Negative: A false negative happens when an element is a member of the set that a Bloom filter represents but the Bloom filter mistakenly reports it is not. A standard Bloom filter has no false negatives. However, in a distributed system, a Bloom filter replica can generate false negatives when the replica is not timely updated.

False Positive: A false positive happens when an element is not a member of the set that a Bloom filter represents but the Bloom filter mistakenly reports it is. The probability of false positives can be very slow when the Bloom filter is appropriately designed.